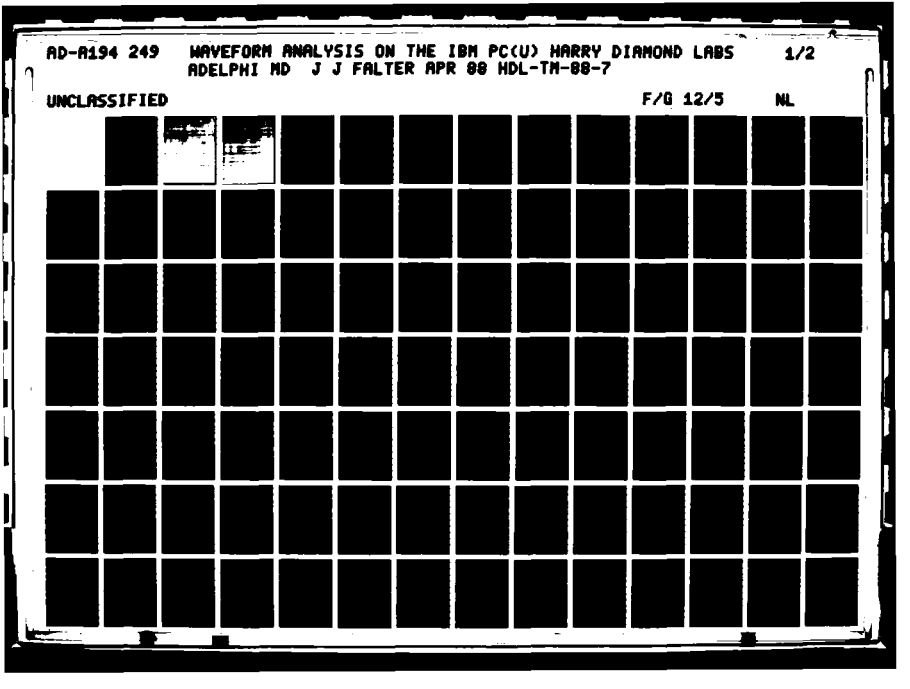
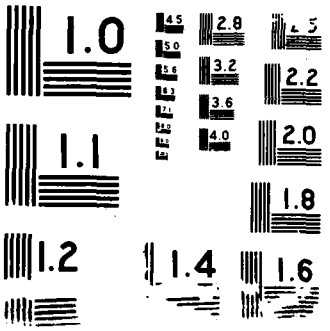
AD-A194 249 WAVEFORM ANALYSIS ON THE IBM PC(U) HARRY DIAMOND LABS 1/2
ADELPHI MD J J FALTER APR 88 HDL-TM-88-7

UNCLASSIFIED F/G 12/5 NL



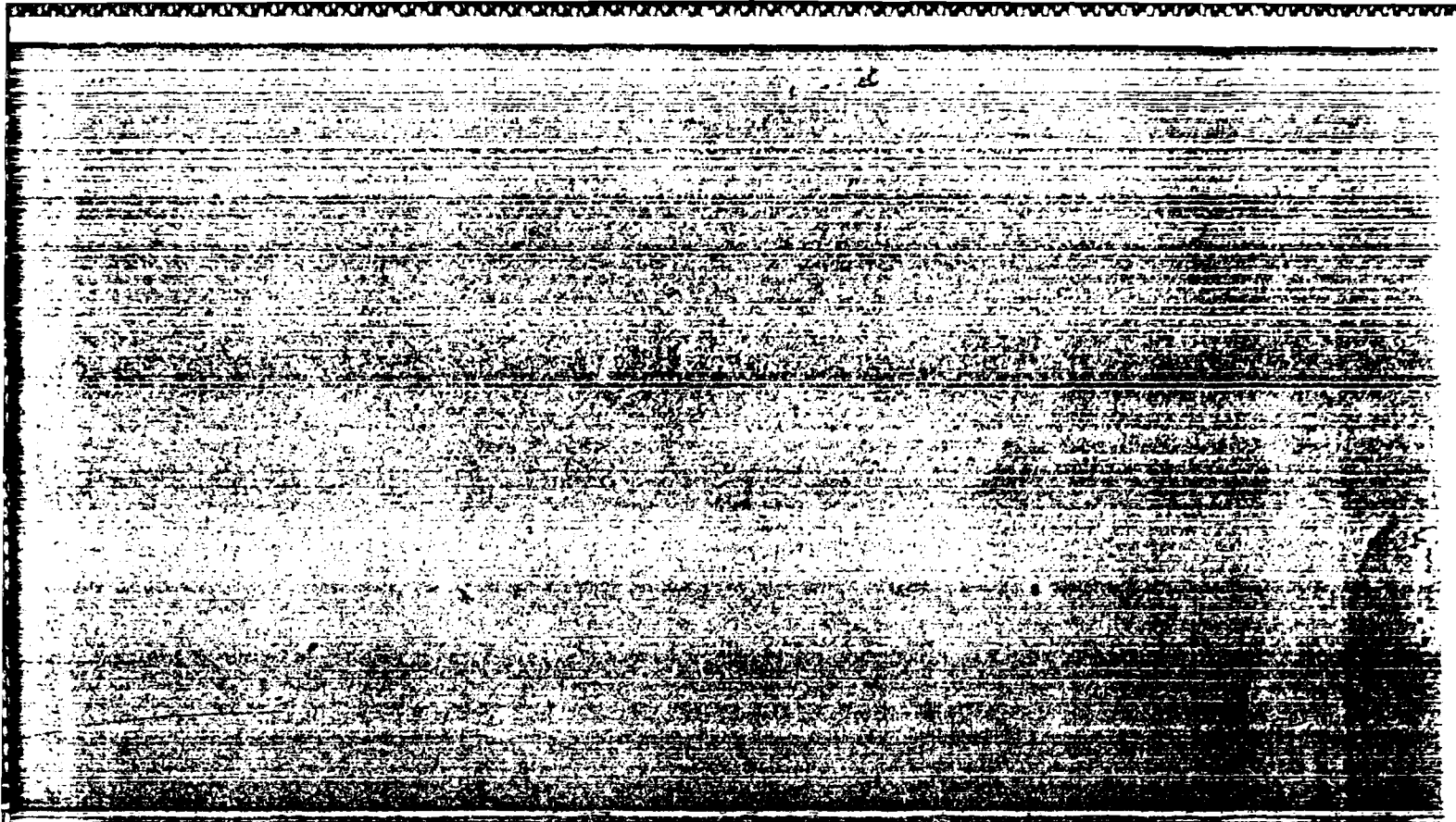

1.0
1.1
1.2
1.4
1.6
2.8
3.2
3.6
4.0
2.2
2.0
1.8

HDL-TM-88-7
April 1988

Waveform Analysis on the IBM PC

by Jeffrey J. Falter

U.S. Army Laboratory Command
Harry Diamond Laboratories
Adelphi, MD 20783-1197



The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturers' or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) HDL-TM-88-7	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Harry Diamond Laboratories 6b OFFICE SYMBOL (If applicable) SLCHD-NW-ED	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) 2800 Powder Mill Road Adelphi, MD 20783-1197	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION U.S. Army Materiel Command 8b OFFICE SYMBOL (If applicable) AMSLC	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) 5001 Eisenhower Avenue Alexandria, VA 22333-0001	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 6.21.20.A; PROJECT NO.; TASK NO.; WORK UNIT ACCESSION NO
11 TITLE (Include Security Classification) Waveform Analysis on the IBM PC	
12 PERSONAL AUTHOR(S) Jeffrey J. Falter	
13a TYPE OF REPORT Final 13b TIME COVERED FROM May 1987 TO Sept 1987 14 DATE OF REPORT (Year, Month, Day) April 1988 15 PAGE COUNT 129	
16 SUPPLEMENTARY NOTATION HDL project: XE77E4; AMS code: 612120.H25	

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Fast Fourier transform (FFT), IBM PC/XT/AT, Turbo Pascal, numerical integration, waveform analysis, numerical differentiation, manual digitization, linear graphs, logarithmic graphs
09	14		
02	03		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes an easy-to-use computer program for the IBM PC/XT/AT to perform data analysis on electromagnetic pulse (EMP) waveforms with minimal computer expertise required. A detailed description of the various algorithms is included, as well as an introductory users' manual. Some of the data analysis options include linear and cubic spline interpolation, fast Fourier transforms (FFT's), inverse FFT's, and numerical integration and differentiation. The program also graphs data waveforms on the screen or on an Epson-compatible printer using linear, logarithmic, or semilog graphs. Additionally, the program reads and writes data files to disk, or data may be manually entered through a digitizing pad. Complete Turbo Pascal program listings are included as appendices.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Jeffrey J. Falter	22b TELEPHONE (Include Area Code) (703) 490-2609 22c OFFICE SYMBOL SLCHD-NW-ED

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Contents

Figures

Algorithms

Tables

1. Introduction

FFT is an easy-to-use computer program for the IBM PC/XT/AT designed to perform data analysis on waveforms. This package was developed at the Harry Diamond Laboratories (HDL) as a tool for analyzing electromagnetic pulse (EMP) waveforms; it can, however, be used for any physically realizable waveforms. Some of the data analysis options include linear and cubic spline interpolation, fast Fourier transforms (FFT's), inverse FFT's, and numerical integration and differentation. The program also graphs data waveforms on the screen or on an Epson-compatible printer using linear, logarithmic, or semilog graphs. Additionally, the program reads and writes data files to disk, or data may be manually entered through a digitizing pad.

The software package FFT was written to provide an easy-to-use method of data analysis for both the experienced and inexperienced computer operator. All functions are menu driven, clearly labelled, and simple to invoke, providing an easy-to-learn and easy-to-use interface to a collection of powerful routines.

2. System Description

2.1 Hardware

The software presented in this report was written for an IBM PC/XT/AT or true compatible, using either a standard color monitor (640 × 200 resolution) or the Hercules graphics adapter and monitor (720 × 350 resolution). A hard disk, while not absolutely necessary, is strongly encouraged. Additionally, the use of the Intel 80x87 math coprocessor is strongly encouraged, since this chip will greatly increase the speed and improve the accuracy of the output of the program (see table 1).

The digitizer used for the manual digitization process is the Micro Digi-Pad.* This is a type 7 absolute electromagnetic 6 by 6 in. digitizer. Other digitizers may be substituted for this one if the appropriate constants are set in the software (e.g., baud rate, number of start and stop bits, and number of data bits).

Table 1. Comparison of real numbers generated with and without an 80x87 math coprocessor.

Feature	With 80x87	Without 80x87
Mantissa	63 bits	40 bits
Characteristic	15 bits	8 bits
Accuracy	19 digits	11 digits
Smallest real	1.9×10^{-4951}	1×10^{-38}
Largest real	1.1×10^{4932}	1×10^{38}

2.2 Software

The software was written in Turbo Pascal version 4.00 (see bibliography)† and was developed under PC-DOS 3.1. Note that these low-level graphics routines were written explicitly for either the Hercules graphics adapter or for one of the IBM color graphics adapters, and hence will not operate with graphics boards which are not compatible with one of these.

The following files must be on the current directory of the logged drive for this program to be used:

FFT.EXE
FFT.cfg (optional configuration file)

**From GTCO Corporation, 1055 First Street, Rockville, MD 20850.*
†Produced by Borland International, 4585 Scotts Valley Drive, Scotts Valley, CA 95066.

In addition, for graphs to be printed, the DOS program GRAPHICS.COM must be able to be run when program FFT is started.

2.3 File Structure

Each data file contains three parts: an identifier, the data points, and an information field. The identifier simply tells the number of data points, n, in a file. The data points are listed in one of two ways (depending on whether the data were transformed or not):

$t_i \; y_i$	(time-domain data)
$t_i \; y_i \; f_i \; m_i \; p_i$	(frequency-domain data)

where

t_i = ith data point,	$1 \le i \le n$,
y_i = amplitude at t_i,	$1 \le i \le n$,
f_i = ith frequency point,	$1 \le i \le n/2$,
m_i = magnitude at f_i,	$1 \le i \le n/2$,
p_i = phase at f_i,	$1 \le i \le n/2$.

The information field contains several information lines. These lines are intended only for reference, and they are just for the convenience of the user. Table 2 is an example of an input data file.

Table 2. Example of an input file.

```
10
0.00        0.00
1.00        0.00
1.01        1.00
3.00        1.00
3.01        0.00
5.00        0.00
5.01        1.00
7.00        1.00
7.01        0.00
8.00        0.00
4s square wave
```

3. Users' Manual

The user starts the program by typing "FFT" at the DOS prompt (e.g., C:>), bringing up the main menu (see fig. 1). Each of the program's several options, described separately below, may be accessed from this menu.

Figure 1. Main menu, entry point of FFT program.

```
        Main Menu  (3.00)
Select Option by typing a number:

         1. Digitizer
         2. Retrieve/Save Data
         3. Signal Processing
         4. Graph Results
         5. Advanced Options
         6. Create Waveform
         9. Exit to System

Your Choice?
```

3.1 Option 1: Digitization

Digitization is selected from the main menu when the user wishes to manually digitize a photo. To begin the digitization process, the photo to be digitized is first placed in the digitizer's drawing area and secured in place. The program asks first for the left and right endpoints of the x-axis, and then for the top and bottom endpoints of the y-axis. Once the coordinate axes have been entered, the user enters (x,y) coordinate pairs, following these general guidelines:

(1) For each point (x_i,y_i), it is necessary that

$$x_{i-1} < x_i < x_r \quad ,$$

where x_r is the right endpoint of the x-axis.

(2) Near points where the curve changes direction, the curve should be sampled more frequently.

(3) Pressing ESCAPE erases an incorrectly entered point.

(4) Pressing ENTER finishes the digitization process.

After all the points are entered, the user is asked to enter the total time and amplitude displayed. These values may be input as simple real numbers (e.g., "0.01"), or as numbers with scale multipliers and/or units (e.g., "10mA"--see table 3). Once these values are entered, the computer redraws the waveform, then waits for a key to be pressed to return to the main menu. (If the results are unsatisfactory, the entire process should be restarted.)

Table 3. Scale multipliers with their real-number equivalents.

Scale multipliers	Value
p	10^{-12}
n	10^{-9}
u	10^{-6}
m	10^{-3}
k	10^{+3}
M	10^{+6}
G	10^{+9}
T	10^{+12}

In program FFT, values may be entered with or without units using the standard metric prefixes (except that μ becomes u). Note that these values are case-dependent; that is, m and M have different values.

3.2 Option 2: File Operations

Selection of option 2 from the main menu will bring up the file I/O menu (see fig. 2). From here the user can save a waveform to disk, retrieve it from disk, or delete it from disk by simply choosing the desired action from the menu choices and typing the filename when prompted. The user may additionally read a disk directory or display the file information lines. The filename can be up to 64 characters long, including a drive and path if desired, in the form d:\full\path\filename.ext. (If either the drive or the path is not specified, then the program will search the current data drive or path, respectively, for the file.) If the program detects an error (for example, disk is full), a brief error message will be displayed.

Figure 2. File I/O menu. All disk input/ output is done from here.

```
            File I/O Menu

  Select Option by typing a number:

          1. Save Data to disk

          2. Retrieve Data from disk

          3. Delete Data from disk

          4. Import DDT file from disk

          5. Disk Directory

          6. Display Information Lines

          9. Exit to Main Menu

  Your choice?
```

3.3 Option 3: Signal Processing

The first time the signal processing functions are called, the data are interpolated, using either cubic spline interpolation (the default) or linear interpolation. Once the interpolation process is complete, the signal processing menu appears (see fig. 3). From this menu, any of several data analysis techniques may be applied.

While most of the functions are self-explanatory, a few remarks must be made:

(1) Signal processing option 3: When an FFT is performed, the prompt "Apply window [H/R/N]?" appears. If the data are nonzero at both the beginning and the end of the waveform, selecting "H" (Hanning window) will force both ends of the data to zero. If the beginning of the data is zero but the end is not, selecting "R" (rear window) will force the end of the data to zero. If both ends of the data are zero, selecting "N" will leave the data unchanged (this is the default).

(2) Signal processing option 5: The transfer function scales a waveform. The scale factor must be of the form

orm

where

o = operation (+, −, *, /)
r = real number (e.g., 10, 0.24, 1e−3)
m = optional multiplier (see table 2).

There should be no spaces in this scale factor. Examples of this function:

$+0.01 \rightarrow f(t) = f(t) + 0.01$
$*1u \rightarrow f(t) = f(t) \times 10^{-6}$

The scale factor can be applied to either axis.

Select signal processing option 9 to return to the main menu.

Figure 3. Signal processing menu. All data analysis is accomplished from here.

```
      Signal Processing Menu
Select Option by typing a number:

        1. Integrate
        2. Differentiate
        3. FFT
        4. Inverse FFT
        5. Transfer Function
        6. Remove DC Level
        9. Exit to Main Menu

Your Choice?
```

3.4 Option 4: Graph Results

This option will graph the current waveform. Graphs may be of one of four types:

1. linear x-axis, linear y-axis,
2. log x-axis, log y-axis,
3. log x-axis, linear y-axis,
4. linear x-axis, log y-axis.

Time-domain data will default to a linear (type 1) graph, the magnitude of frequency-domain data will default to a log (type 2) graph, and the phase of the frequency-domain data will default to a log-linear (type 3) graph. (Phase can only be plotted on a linear or log-linear graph.) If an FFT has been performed on the data, the prompt "Time, Magnitude, or Phase [T/M/P]?" will appear. (Time and magnitude are the most common displays; phase is included only for completeness.)

The program graphs the waveform in the upper two thirds of the screen. The user may then opt to print the graph, with a title and subtitle, or change either the type of plot or the interval boundaries of the plot, from the prompts on screen. When finished viewing the graph, the user may return to the main menu by selecting option 9.

3.5 Option 5: Advanced Options

Advanced options allow the user to change several default values of the program (see fig. 4). Defaults that may be changed are the type of interpolation to be performed, to which port the digitizer is connected, whether a printer is installed or not, the background and foreground screen colors (on color monitors), and the data disk/directory. Options can be saved. Saved options are read at program startup. All changes are made from the advanced options menu.

On the menu, the user is asked to choose the digitizer port: "[0/1/2]?" Here, "0" refers to no digitizer, "1" to a digitizer on COM1, and "2" to a digitizer on COM2. Type in the appropriate number.

3.6 Example Session

The following example assumes that a waveform is on file, but you could also have created a waveform using the create waveform menu (see fig. 5). This menu is accessed from the main menu (fig. 1), and is used to create general mathematical and analytical waveforms.

In this example, we assume that there is a 4-MHz sine wave in file SINE.RAW on drive A. You want to perform an FFT and then an inverse FFT on the wave, and store the results in file SINE.FFT on drive A. From the DOS prompt (C:>) type FFT to enter the data analysis program. This brings you to the main menu (see fig. 1). Choose option 5 (advanced op-

Figure 4. Advanced options menu. All system parameters are modified here.

```
        Advanced Options
Select Option by typing a number:

     1. Linear/Spline Interpolation:    S
     2. Digitizer Port [0/1/2]:         1
     3. Printer installed (y/n) ?       N
     4. Colors:                    10/ 1/ 7
     5. Data Disk/Directory:           c:
     6. Save Options
     9. Exit to Main Menu


Your Choice?
```

Figure 5. Create waveform menu. All analytic waveforms are created from this menu.

```
        Create Waveform Menu
Select waveform to create by typing a number.

        1. Damped Sine Wave
        2. EMP Double Exponential
        3. Reciprocal Double Exponential
        4. General Exponential Sum
        9. Exit to Main Menu


Your Choice?
```

tions) to get to the advanced options menu, then option 5 to change the default data disk directory. Type A:<ENTER>. Next, choose option 9 to return to the main menu, then choose option 2 (retrieve/save data) to get to the file I/O menu (see fig. 2), then option 2 (retrieve data from disk). When prompted for a filename, type A:SINE.RAW <ENTER>. After the file is read from the disk, you are returned to the main menu.

Before performing an FFT on the data, take a look at its graph by choosing option 4 (graph results) from the main menu. Choosing option 1 (print screen) gives you the graph in figure 6.

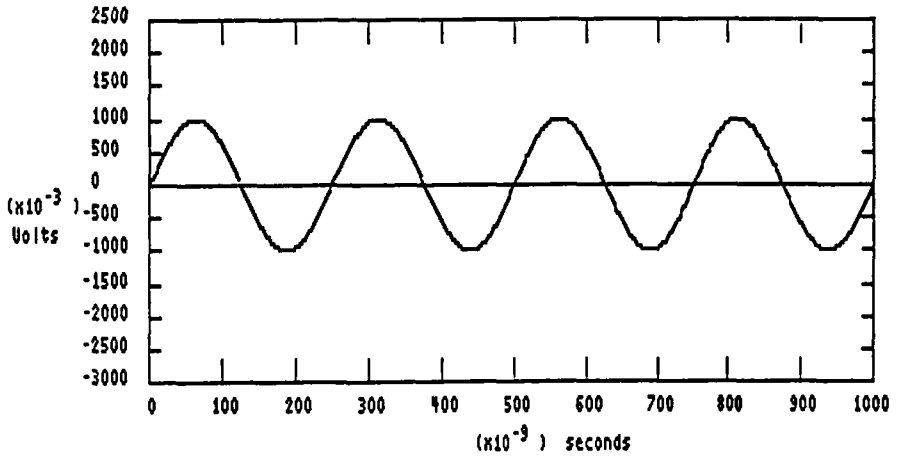


Figure 6. A 4-MHz sine wave in time domain.

Next choose option 9 to return to the main menu, then option 3 (signal processing) to get to the signal processing menu (see fig. 3). From the signal processing menu, choose option 3 (FFT) to perform an FFT on the waveform. Since the sine wave both starts and ends at zero, no window need be applied, so answer "N" to the window prompt.

When the FFT is done, choose option 9 to return to the main menu, then option 4 (graph results) to see the results of the transform. Answer the "Time, Magnitude, or Phase?" prompt with an "M" to see the magnitude of the frequency-domain data. Once the graph is drawn, choosing option 1 (print screen) gives the plot of figure 7.

Often it is desirable to view the graph on a linear scale. To do this, choose option 2 (change parameters) and then choose type 1 (linear x-axis, linear y-axis). When the graph is redrawn and printed using option 1 (print screen), the plot of figure 8 is obtained.

Tc perform an inverse FFT, return to the main menu by selecting option 9, then return to the signal processing menu by selecting option 3. From the signal processing menu, choose option 4 (inverse FFT). When the inverse FFT is finished, return to the main menu, then choose option 4 (graph results), this time selecting "T" at the prompt "Time, Magnitude, or Phase?" This displays the time-domain function as a result of an FFT followed by an inverse FFT. The result, when printed, is figure 9.

Now to save the data, choose option 9 to return to the main menu, then option 2 (retrieve/save data) to reach the file I/O menu. Choose option 1 (save data to disk) and, when prompted, save the wave to file A: SINE.FFT. When returned to the main menu, choose option 9 (exit to system) to leave the program. You are done.

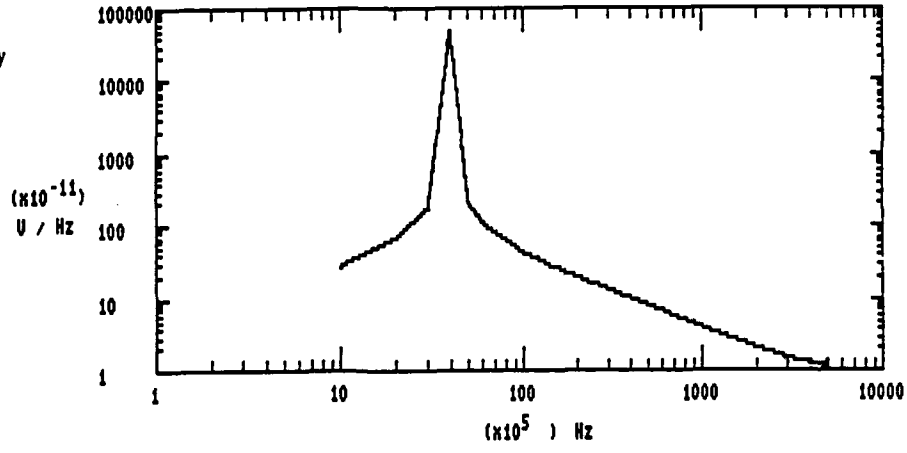


Figure 7. A 4-MHz sine wave in frequency domain (magnitude part only).

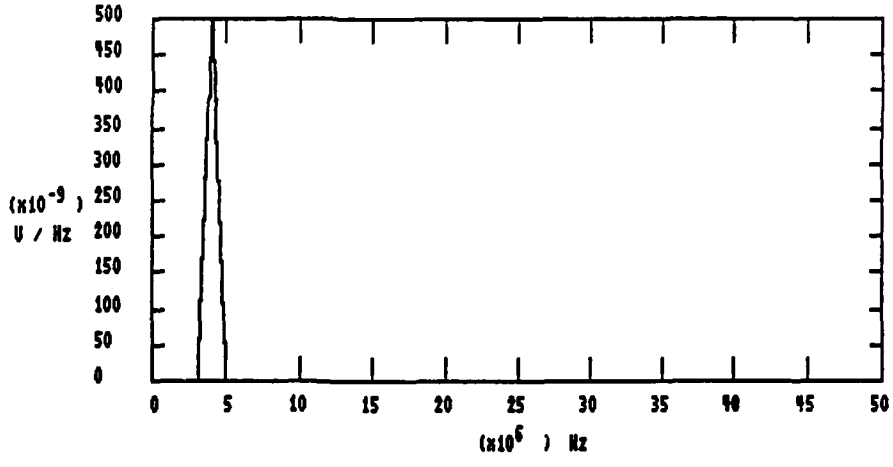
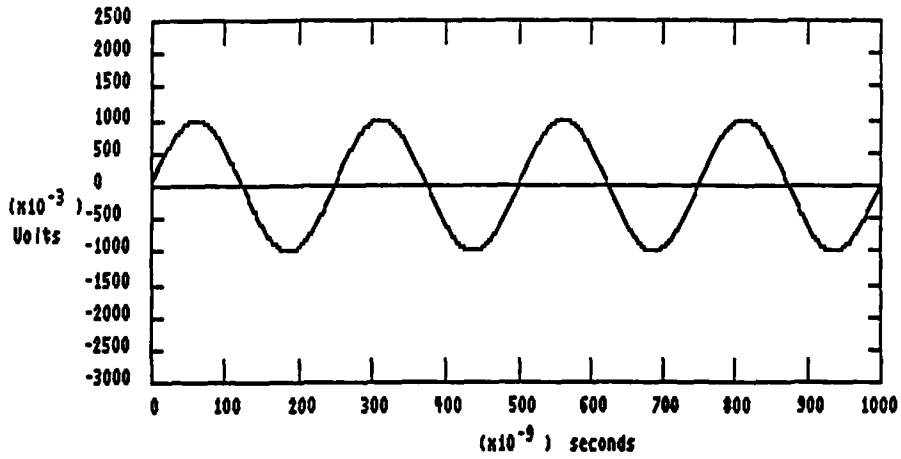


Figure 8. An alternate representation of a 4-MHz sine wave in frequency domain.

Figure 9. A 4-MHz sine wave in time domain after a forward and then inverse FFT.



4. Mathematical Theory

Each of the main mathematical functions of the program is discussed in detail below. The implementation of these functions can be found in the program listings of the appendix.

4.1 Rotation

A digitized waveform is input with respect to the horizontal and vertical axes of the oscilloscope photograph (oscillograph). These axes, however, may not be truly horizontal and vertical with respect to the digitizer's coordinate system and, in fact, these axes may not even be perpendicular to each other. (Such a coordinate system is called oblique.) It may be necessary, therefore, to rotate the digitized waveform (via software) in order to align its coordinate system with the digitizer's coordinate system (see fig. 10). Procedure Rotate_Waveform, called automatically after the photo has been digitized, performs this rotation in three steps:

(1) Calculation of the "true" origin, (x_0, y_0).

(2) Conversion of the oblique coordinates to rectangular components, (x_i, y_i).

(3) Calculation of the scaling factors, x-factor and y-factor.

Calculation of the "true" origin is necessary since the x- and y-axes, as input, may not intersect. The true origin, (x_0, y_0), is obtained from the formulas

$$m_x = \frac{y_2 - y_1}{x_2 - x_1} \ ,$$

$$m_y = \frac{y_4 - y_3}{x_4 - x_3} \ ,$$

$$x_0 = \frac{y_2 - y_3 + m_y x_3 - m_x x_2}{m_y - m_x} \ ,$$

$$y_0 = y_2 + m_x (x_0 - x_2) \ ,$$

where

m_x = the slope of the digitized x-axis,

m_y = the slope of the digitized y-axis,

x_0 = the x-coordinate of the origin, and

y_0 = the y-coordinate of the origin.

(Note that if $x_3 = x_4$, it is necessary that $x_0 = x_3$.) These formulas come from the equations

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \text{ and}$$

$$y - y_4 = \frac{y_3 - y_4}{x_3 - x_4}(x - x_4)$$

by substituting (x_0, y_0) for (x, y).

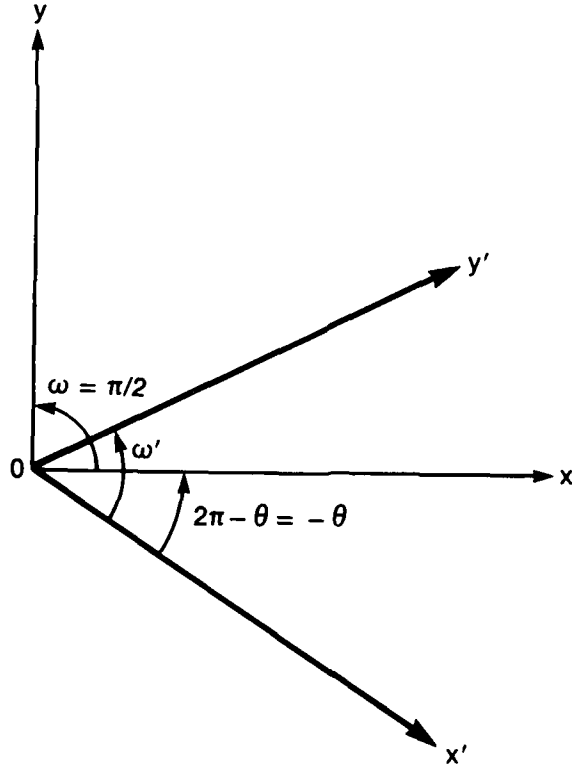


Figure 10. Input waveform before rotation (x', y') and after rotation (x, y).

Conversion of the input (oblique) coordinates to rectangular coordinates is accomplished by the formulas

$$x = x' \cos(\theta) + y' \cos(\omega' + \theta) \text{ and}$$

$$y = x' \sin(\theta) + y' \sin(\omega' + \theta) \ .$$

These formulas come from the equations (from CRC *Standard Math Tables*, 1984)

$$x = \frac{x' \sin(\omega - \theta) + y' \sin[\omega - (\omega' + \theta)]}{\sin \omega} \text{ and}$$

$$y = \frac{x' \sin(\theta) + y' \sin(\omega' + \theta)}{\sin \omega}$$

with

$$\omega = \frac{\pi}{2} \text{ and}$$

$$\sin\left(\frac{\pi}{2} \pm \alpha\right) = \cos \alpha \ .$$

Lastly, the digitizer coordinates are scaled to the true coordinates as follows. The length of each axis is calculated:

$$r_x = \sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2} \ ,$$

$$r_y = \sqrt{(x_3 - x_4)^2 + (y_3 - y_4)^2} \ .$$

The user then inputs the total time and total amplitude displayed in the photo. The scaling factors are then

$$x\text{-factor} = (\text{total time})/r_x \ ,$$

$$y\text{-factor} = (\text{total amplitude})/r_y \ .$$

The "true" coordinates are then the digitizer coordinates times the scaling factor.

If the photo's coordinate system were not rotated into the digitizer's coordinate system, a certain amount of error would occur. This error, which would depend on the angle between the axes of the two coordinate systems, θ, would be proportional to $\cos \theta$.

4.2 Interpolation

Input data, whether from digitizer or disk, may contain an arbitrary number of data points, taken at arbitrary time intervals. In order to perform any of the signal processing functions, we must transform this randomly spaced data to a collection of equally spaced data points which still accurately represent the original data. In addition, the FFT algorithm is a "power of 2" algorithm which requires 2^n input points (for some integer n), so the arbitrary number of points in the input waveform must be interpolated into 2^n points. This interpolation of the data is performed automatically before any of the signal processing functions are called the first time. Interpolation is accomplished either by procedure LinearInterpolation, a modification of subroutine LINT by Noon (1976) (see bibliography), or by procedure CubicSpline, a modification of the natural cubic spline algorithm by Burden and Faires (1985).

4.2.1 Linear Interpolation

Linear interpolation joins the set of data points $\{(x_0,y_0), (x_1,y_1), ..., (x_n,y_n)\}$ by a series of straight lines. A data point (x,y) is then interpolated between two given points by calculating the equation of the line through the points, then using the desired value of x to calculate the corresponding value of y (see fig. 11). The equations for this are

$$x = x_i + h \ ,$$

$$y = \left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) x + \frac{y_i x_{i+1} - y_{i+1} x_i}{x_{i+1} - x_i} \ .$$

The disadvantage of using linear interpolation is that at each of the endpoints of the subintervals there is no assurance of differentiability, meaning that the interpolated function may not be smooth at these points. To obtain a continuously differentiable (and hence smooth) interpolated function, a natural cubic spline can be applied to the data.

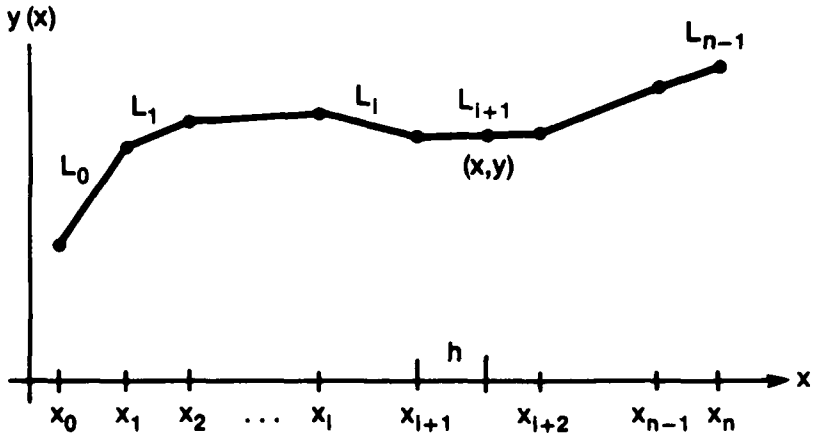


Figure 11. Linear interpolation of point (x,y) between points (x_{i+1}, y_{i+1}) and (x_{i+2}, y_{i+2}).

4.2.2 Cubic Spline Interpolation

Definition: Given a set of data points $\{(x_0,y_0), (x_1,y_1), ..., (x_n,y_n)\}$ such that $x_0 < x_1 < ... < x_n$, a natural cubic spline interpolant, S, satisfies the following conditions:

(1) S is a cubic polynomial, denoted S_j, on the subinterval $[x_j, x_{j+1}]$ for each $j = 0, 1, ..., n-1$, such that $S_j = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$ where $x_j \le x < x_{j+1}$,

(2) $S(x_j) = y_j$, for each $j = 0, ..., n$,

(3) $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$, for each $j = 0, ..., n-2$,

(4) $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$, for each $j = 0, ..., n-2$,

(5) $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$, for each $j = 0, ..., n-2$,

(6) $S''(x_0) = S''(x_n) = 0$.

From the definition, it is seen that not only will the interpolated function S be smooth, but so too will the first and second derivatives of S (Burden and Faires (1985), pp 117-129).

Procedure CubicSpline (see algorithm 1) first calculates the coefficients a_j, b_j, c_j, d_j of each S_j. Next, after all these coefficients have been found, a data point is interpolated between two given points by using the desired

value of x and the appropriate S_j to calculate the corresponding value of y (see fig. 12). The equations for this are

$$x = x_j + h \ , \qquad \text{where } x_j \le x < x_{j+1} \ ,$$

$$y = a_j + b_j h + c_j h^2 + d_j h^3 \ .$$

Most EMP waveforms can be accurately interpolated using the cubic spline procedure (for example, compare fig. 13 and 14), and hence this is the default interpolation method. The operator has the option, however, of using linear interpolation instead. This is useful since some waveforms, particularly those containing sharp discontinuities (such as the square wave of fig. 15), may be much more accurately interpolated by the linear procedure (compare fig. 16 and 17). Once interpolation has been performed, any of the data processing functions may be called.

4.3 Integration

Integration is performed on the data $\{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$ using the iterated trapezoidal rule, as given by Tektronix (1985):

$$y_0 = 0 \ ,$$

$$y_j = y_{j-1} + \frac{h[f(x_{j-1}) + f(x_j)}{2} \ ,$$

where

$f \in C^2[a,b]$,

$$h = \frac{b-a}{n}$$

$x_j = a + jh$, for $0 \le j \le n$, and j is an integer.

This routine has an error of $O(h^2)$.

Algorithm 1. Natural cubic spline algorithm (Burden and Faires, 1985).

To construct the cubic spline interpolant S for the function f, defined at the numbers $x_0 < x_1 < \cdots < x_n$, satisfying $S''(x_0) = S''(x_n) = 0$:

INPUT n; $x_0, x_1, \ldots, x_n$; either generate $a_i = f(x_i)$ for $i = 0, 1, \ldots, n$ or input a_i for $i = 0, 1, \ldots, n$.

OUTPUT a_j, b_j, c_j, d_j for $j = 0, 1, \ldots, n-1$.
(*Note*: $S(x) = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$ *for* $x_j \le x < x_{j+1}$.)

Step 1 For $i = 0, 1, \ldots, n-1$ set $h_i = x_{i+1} - x_i$.

Step 2 For $i = 1, 2, \ldots, n-1$ set

$$z_i = \frac{3[a_{i+1}h_{i-1} - a_i(x_{i+1} - x_{i-1}) + a_{i-1}h_i]}{h_{i-1}h_i}.$$

Step 3 Set $l_0 = 1$; (*Steps 3, 4, 5, and part of Step 6 solve a tridiagonal linear system using Algorithm 6.7.*)
$\mu_0 = 0$;
$z_0 = 0$.

Step 4 For $i = 1, 2, \ldots, n-1$
set $l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$;
$\mu_i = h_i/l_i$;
$z_i = (z_i - h_{i-1}z_{i-1})/l_i$.

Step 5 Set $l_n = 1$;
$z_n = 0$;
$c_n = 0$.

Step 6 For $j = n-1, n-2, \ldots, 0$
set $c_j = z_j - \mu_j c_{j+1}$;
$b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$;
$d_j = (c_{j+1} - c_j)/(3h_j)$.

Step 7 OUTPUT (a_j, b_j, c_j, d_j *for* $j = 0, 1, \ldots, n-1$);
STOP.

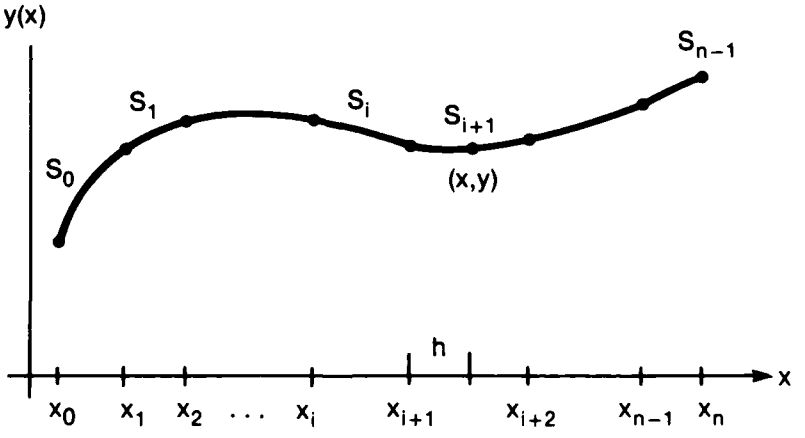


Figure 12. Cubic spline interpolation of point (x, y) between points (x_{i+1}, y_{i+1}) and (x_{i+2}, y_{i+2}).

Figure 13. Double exponential before interpolation.

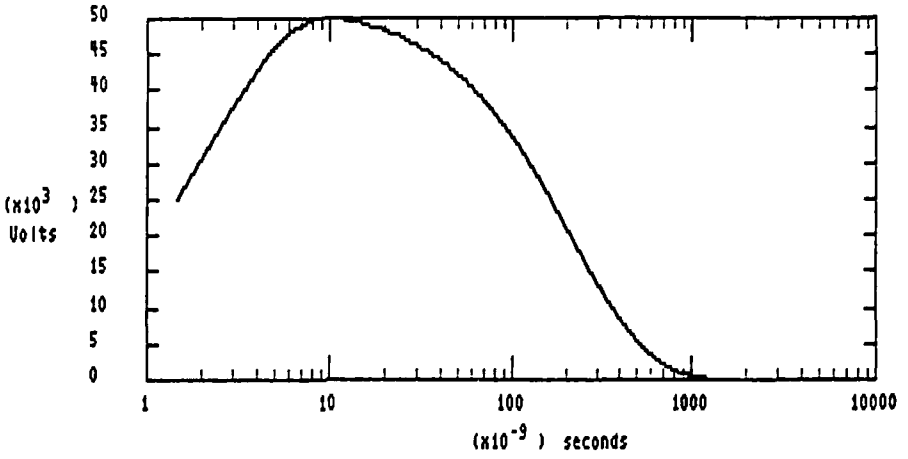


Figure 14. Double exponential using cubic spline interpolation.

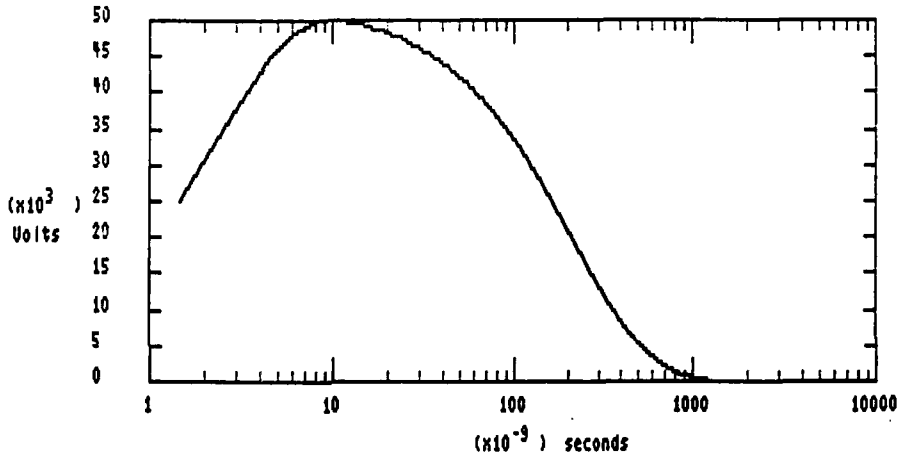
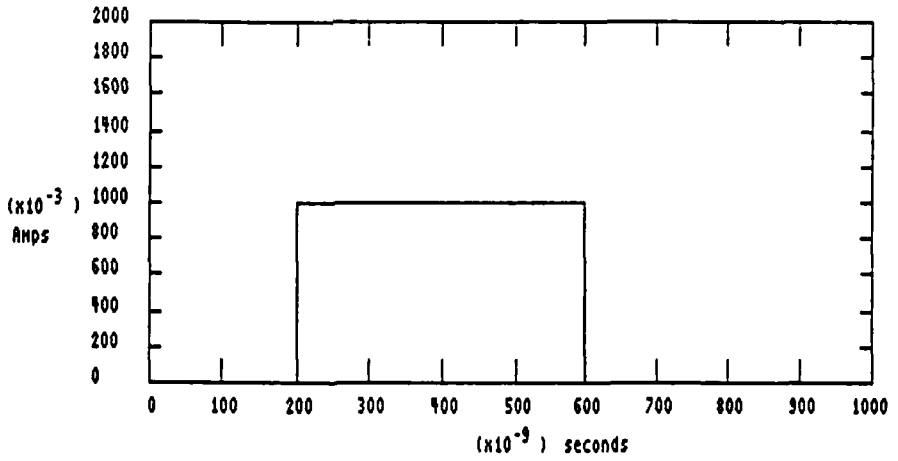


Figure 15. Square wave before interpolation.



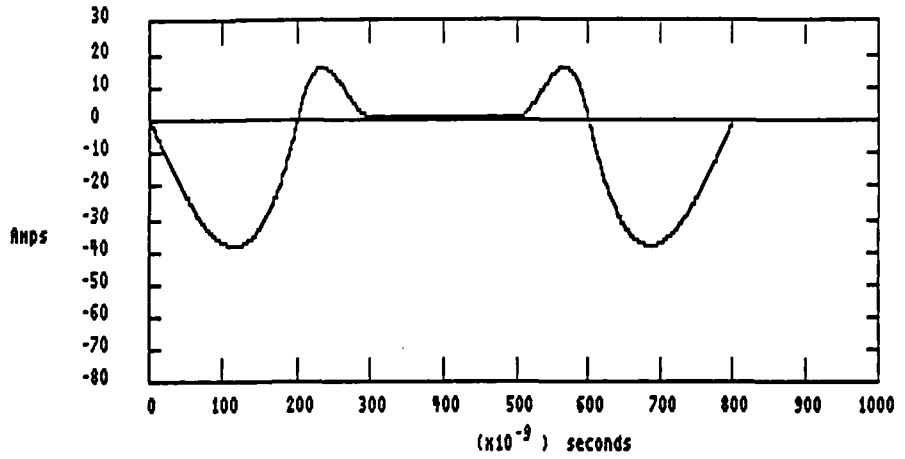


Figure 16. Square wave using cubic spline interpolation.

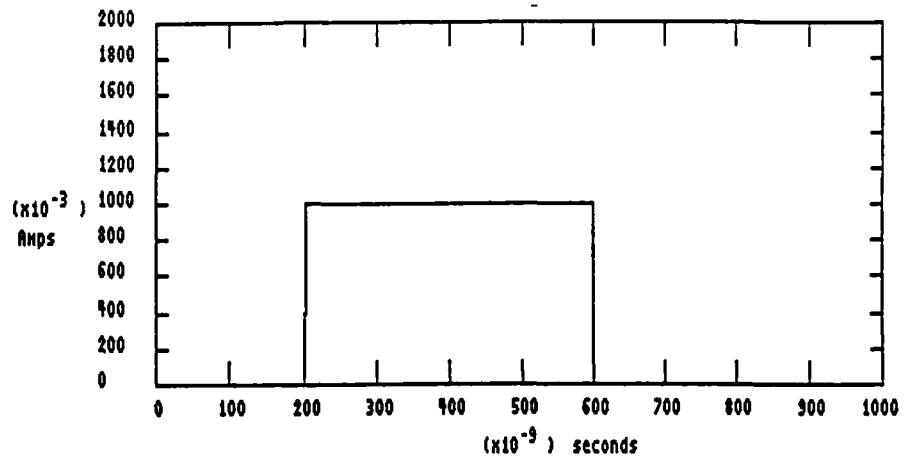


Figure 17. Square wave using linear Interpolation.

4.4 Differentiation

Differentiation is accomplished on the data $\{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$ via the three point formulas, as given in Burden and Faires (1985, pp 137-145). These formulas are

$$f'(x_j) = \frac{f(x_j + h) - f(x_j - h)}{2h} \quad ,$$

$$f'(x_0) = \frac{-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)}{2h} \quad ,$$

$$f'(x_n) = \frac{f(x_n - 2h) - 4f(x_n - h) + 3f(x_n)}{2h} \quad ,$$

where h and x_j are as defined above for the integration rule. This routine also has an error of $O(h^2)$.

4.5 Fast Fourier Transform

Much of the discussion of windows in this section closely follows the work of Shreve (unpublished).

The FFT is performed by a modification of the Cooley-Tukey algorithm, as given in Burden and Faires (1985, pp 398-406) (see algorithm 2). Note that the FFT operates on a finite interval of data, treating the data as if they were periodic. It is therefore necessary that the beginning of the waveform be continuous with the end of the waveform; in other words, the data should

(1) start and end at 0 (or some other constant), and

(2) approach this value at the endpoints of the waveform asymptotically, so as to avoid introducing discontinuities.

While the algorithm will still function if these two conditions are not met, the results may not be as accurate.

As suggested by Shreve (unpublished), if a waveform on an interval T is nonzero at one or both of the endpoints, it may be forced to zero through the use of a window. (While any windowing function will introduce distortion into the transform, this distortion is often less than the distortion introduced by a discontinuity in the waveform, such as will occur if the above two conditions are not satisfied.) Two types of windows may be applied: a Hanning window or a rear window. If neither end of the data is zero, then a Hanning (or cosine) window will apply the function

$$f(t) = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi t}{T}\right)\right], \qquad 0 \le t \le T,$$

to the waveform. This function is zero at both endpoints, and will force the waveform to follow. If the data at the beginning of the sampled waveform are equal to zero but the data at the end are not, then a rear window will apply the function

$$f(t) = \frac{1}{2}\left\{1 - \cos\left[\frac{\pi(t+T)}{T}\right]\right\}, \qquad 0 \le t \le T,$$

Algorithm 2. Cooley-Tukey FFT algorithm (Burden and Faires, 1985).

To compute the discrete approximation

$$F(x) = \frac{1}{m}\sum_{k=0}^{2m-1} c_k e^{ikx} = \frac{1}{m}\sum_{k=0}^{2m-1} c_k(\cos kx + i \sin kx) \qquad \text{where } i = \sqrt{-1},$$

for the data $\{(x_j, y_j)\}_{j=0}^{2m-1}$ where $m = 2^p$ and $x_j = -\pi + j\pi/m$ for $j = 0, 1, \ldots, 2m-1$

INPUT $m, p; y_0, y_1, \ldots, y_{2m-1}$.

OUTPUT complex numbers $c_0, \ldots, c_{2m-1}$; real numbers $a_0, \ldots, a_m; b_1, \ldots, b_{m-1}$.

Step 1 Set $M = m$;
$q = p$;
$\zeta = e^{\pi i/m}$.

Step 2 For $j = 0, 1, \ldots, 2m-1$ set $c_j = y_j$.

Step 3 For $j = 1, 2, \ldots, M$ set $\xi_j = \zeta^j$;
$\xi_{j+M} = -\xi_j$.

Step 4 Set $K = 0$;
$\xi_0 = 1$.

Step 5 For $L = 1, 2, \ldots, p+1$ do Steps 6–12.

Step 6 While $K < 2m-1$ do Steps 7–11.

Step 7 For $j = 1, 2, \ldots, M$ do Steps 8–10.

Step 8 Let $K = k_p \cdot 2^p + k_{p-1} \cdot 2^{p-1} + \cdots + k_1 2 + k_0$; (*Decompose k.*)
set $K_1 = K/2^q = k_p \cdot 2^{p-q} + \cdots + k_{q+1} \cdot 2 + k_q$;
$K_2 = k_q \cdot 2^p + k_{q+1} \cdot 2^{p-1} + \cdots + k_p \cdot 2^q$.

Step 9 Set $\eta = c_{K+M}\xi_{K_2}$;
$c_{K+M} = c_K - \eta$;
$c_K = c_K + \eta$.

Step 10 Set $K = K + 1$.

Step 11 Set $K = K + M$.

Step 12 Set $K = 0$;
$M = M/2$;
$q = q - 1$.

Step 13 While $K < 2m-1$ do Steps 14–16.

Step 14 Let $K = k_p \cdot 2^p + k_{p-1} \cdot 2^{p-1} + \cdots + k_1 \cdot 2 + k_0$; (*Decompose k.*)
set $j = k_0 \cdot 2^p + k_1 \cdot 2^{p-1} + \cdots + k_{p-1} \cdot 2 + k_p$.

Step 15 If $j > K$ then interchange c_j and c_k.

Step 16 Set $K = K + 1$.

Step 17 Set $a_0 = c_0/m$;
$a_m = \text{Re}(e^{-m\pi i}c_m/m)$.

Step 18 For $j = 1, \ldots, m-1$ set $a_j = \text{Re}(e^{-j\pi i}c_j/m)$;
$b_j = \text{Im}(e^{-j\pi i}c_j/m)$.

Step 19 OUTPUT $(c_0, \ldots, c_{2m-1}; a_0, \ldots, a_m; b_1, \ldots, b_{m-1})$;
STOP.

to the waveform. This is the last half of the Hanning window applied over the whole interval of the waveform. It is up to the user to choose the correct window for the data.

"Both [the Hanning and Rear] windows introduce distortion into the transform, but often this distortion is far less objectionable than the 'distortion' introduced by the discontinuities" (Shreve, unpublished). Effectively, the windows convolve a pair of functions with the transform. These functions are half-amplitude $\sin(x)/x$ functions with the rear window, and half-amplitude single sidebands with the Hanning window.

4.6 Remove Mean

The mean of the time-domain waveform may be removed from the waveform. The mean is calculated as the sum of the magnitudes divided by the number of points:

$$\text{mean} = \frac{1}{n} \sum_{i=1}^{n} y_n \quad ,$$

Then

$$y_i = y_i - \text{mean} \, , \quad \text{for } 0 \le i \le n \quad .$$

Acknowledgements

I wish to gratefully acknowledge the support and contribution of Bob Atkinson in the development of the computer software presented in this report. This endeavor would have been impossible without his assistance. I also wish to thank Jim Loftus for his assistance in debugging the FFT routine, and for the many helpful comments he has provided.

Bibliography

Atkinson, Robert L., *Digitizing Waveforms with Prodesign II*, Harry Diamond Laboratories, HDL-TM-87-6 (August 1987).

Burden, Richard L., and J. Douglas Faires, *Numerical Analysis*, 3rd ed., Prindle, Weber, and Schmidt, Boston, MA (1985); *Cubic Spline Interpolation*, pp 117-129; *Numerical Differentiation*, pp 137-145; *Composite Numerical Integration*, pp 162-167; *Trigonometric Polynomial Interpolation*, pp 398-406.

CRC Press, *CRC Standard Math Tables*, 27th ed., Boca Raton, FL (1984).

Elliot, Douglas F., and K. Ramamohan Rao, *Fast Transforms--Algorithms, Analyses, Applications*, Academic Press, Inc., Orlando, FL (1982).

Noon, Thomas V., *Users' Manual for the Modular Analysis Package Libraries ANAPAC and TRANL*, Harry Diamond Laboratories, HDL-TR-1782 (November 1976).

Ramirez, Robert W., *The FFT--Fundamentals and Concepts*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1985).

Shreve, James S., *Instruction Manual for Data Reduction 10*, Harry Diamond Laboratories, unpublished.

Stein, Philip, *How is the Logarithmic Scale Produced?*, in *Graphical Analysis--Understanding Graphs and Curves in Technology*, Hayden Book Co., Inc. New York, NY (1964), pp 153-154.

Tektronix, Inc., *Signal Processing Package*, TEK SPS Basic V03XM System Software (1985), pp 1.19-1.23.

Turbo Pascal (version 4.00) Reference Manual, Borland International, Inc., Scotts Valley, CA (1987).

Turbo Graphix Toolbox (version 1.05A), Borland International, Inc., Scotts Valley, CA (1985).

The Turbo Pascal Tutor, Borland International, Inc., Scotts Valley, CA (1984), pp 20.1-20.27.

Turbo Pascal Numerical Methods Toolbox, Borland International, Inc., Scotts Valley, CA (1986).

APPENDIX A. LISTING FOR SOFTWARE PACKAGE FFT

APPENDIX A

Contents

Illustrations

A-1. Why Pascal

Why Pascal? A common question. Pascal is often referred to as a non-scientific language, not for use in "real" scientific programming. Why, then, do I choose to use it for FFT? The primary reasons are

(1) the speed of compilation of the Turbo Pascal compiler (up to 27,000 lines per minute);

(2) the speed of execution of the resulting programs;

(3) the strong type-checking of Pascal, which eliminates many of the insidious errors that so often plague FORTAN and BASIC programs;

(4) the advanced data structures that are found in Pascal which are not found in FORTRAN and BASIC, and which make programming immensely easier and more logical;

(5) the readability and the ease of debugging of the resulting code, compared to FORTRAN or BASIC; and

(6) the simple elegance of Pascal versus FORTRAN or BASIC.

While some may say that the advantages of Pascal are purely "cosmetic" compared to the advantage of the huge libraries available to FORTRAN programmers, I have found that my productivity has improved since starting to use Pascal last year, since less time is spent waiting for the compiler to do its job, and less time is wasted on stupid errors.

A-2. Availability of Program

Anyone possessing Turbo Pascal 4.0 and who has the necessary hardware (see table A-1) should be able to type in program FFT, then compile and run it, but typing 3000 lines of code can be, to say the least, a tedious task. Therefore, a limited number of copies can be made available to interested readers who contact the author at the number given on p 1 of this report.

APPENDIX A

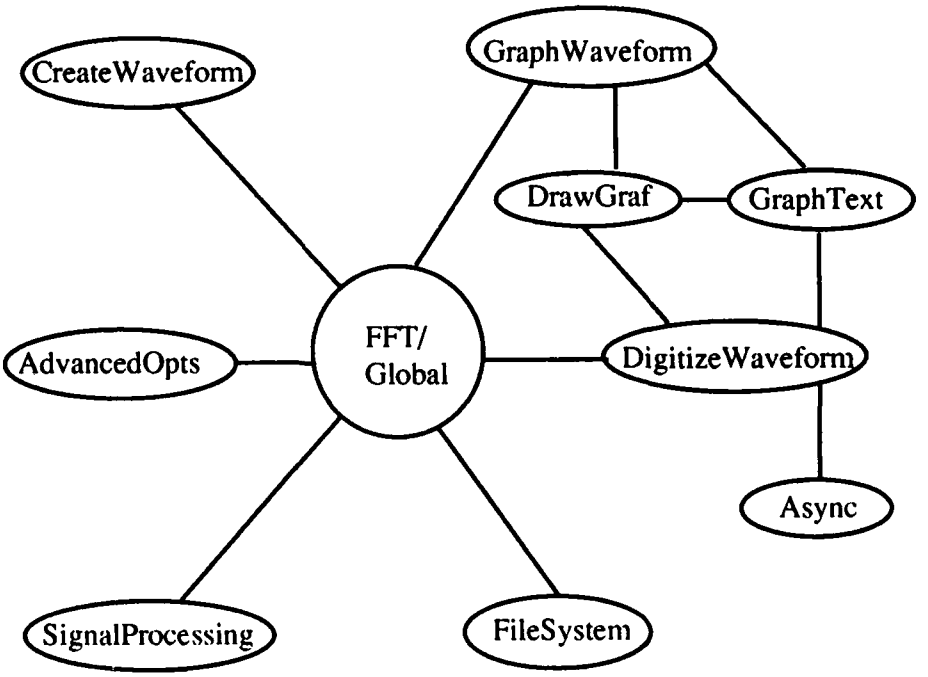
Table A-1. Hardware needed to run FFT.

Component	Type
Computer	IBM PC/XT/AT or compatible
Operating system	PC/MS-DOS 3.0 or higher
Minimum memory	640K (will not use EMS/EEMS)
Graphics hardware	Hercules monochrome graphics CGA, EGA, or VGA
Digitizer	GTCO Corp. Micro Digi-Pad
Floating-point hardware	80x87 (strongly recommended)

A-3. Program Listings

The listings that follow begin with the main program, FFT, and continue with the various subroutines. Figure A-1 shows the interrelationships of the main program and its subroutines.

Figure A-1. Inter-relationship of units of FFT.



A-3.1 FFT

File FFT contains the main calling routines for program FFT. All supporting files are loaded, the system is initialized, and control is passed to various procedures, as selected by the user from the main menu. The file is structured as follows:

1. Comments.

2. Include separate units.

3. Main program
   a. Initialize system
   b. Repeat

      (1) Option 1: Digitize Data
      (2) Option 2: Disk File Operations
      (3) Option 3: Data Analysis
      (4) Option 4: Graph Results
      (5) Option 5: Advanced Options
      (6) Option 6: Create Waveform
      (7) Option 9: Finished

      until finished

   c. End of program

APPENDIX A

```
(*****************************************************************************)
(*                                                                           *)
(* Program written by Jeff Falter, SLCHD-NW-ED.                              *)
(*                                                                           *)
(*****************************************************************************)
(*                                                                           *)
(* To change the number of points which may be operated upon, the following *)
(* constant(s) must be changed in file Global:                               *)
(*     1. TNArraySize : the maximum number of input points allowed.          *)
(*                                                                           *)
(*****************************************************************************)


Program FFT;

uses
   Crt,
   Graph,
   Global,
   DigitizeWaveform,
   FileSystem,
   SignalProcessing,
   GraphWaveform,
   AdvancedOpts,
   CreateWaveform;


(*****************************************************************************)

begin   {FFT}
   (*** First initialize variables. ***)
   GraphicsCard:=Detect;
   DetectGraph (GraphicsCard,GraphicsMode);
   Read_DefaultOptions;
   TextColor (ForeColor);
   TextBackground (BackColor);
   ClrScr;
   Initialize_Variables;

   (*** Create windows, show menu, ask choice. ***)
   repeat
      MainMenu;
      case MainChoice of                    (* Choices:                    *)
         1: begin   {create}                (* 1. Data from digitizer      *)
               if (SerialPort <> 0) then begin
                  PrintText ('Digitize time or frequency data (T/F)? ',
                             StartColumn-1,23);
                  repeat
                     MainChar:=UpCase(ReadKey);
                  until (MainChar in ['T','F',ENTER]);
                  if (MainChar = ENTER)
                     then Digitizer ('T')
                     else Digitizer (MainChar);
               end;   (if)
            end;   {create}
         2: FileIO;                         (* 2. File operations.         *)
         3: if ORIG OR TRANS then begin
               if NOT ACCEPT then
                  PrintText ('Interpolation in progress. Please wait ... ',
                             StartColumn-1,23);
               SPS;                         (* 3. Signal Processing.       *)
            end;
```

```
            4: begin   {graph}                (* 4. Graph data.              *)
                  if TRANS
                     then begin
                        PrintText ('Time, Magnitude or Phase (T/M/P)?',
                                   StartColumn-1,23);
                        repeat
                           MainChar:=UpCase(ReadKey);
                        until (MainChar in ['T','M','P',ENTER]);
                        if (MainChar = ENTER)
                           then if ORIG
                             then GraphResults ('T')
                             else GraphResults ('M')
                           else GraphResults (MainChar);
                     end   {then}
                  else if ORIG then GraphResults ('T');
               end;   {graph}
            5: AdvancedOptions;               (* 5. Advanced Options         *)
            6: begin
                  AnalyticWaveform;           (* 6. Create Analytic Waveform. *)
                  if ORIG then begin
                     GraphResults ('T');
                  end;   {if}
               end;
            9: begin   {finished}             (* 9. Exit program.            *)
                  DONE:=true;
                  ClrScr;
                  Release (HeapTop);
               end;   {finished}
         end;   {case}
      until DONE;
   end.   {FFT}
```

A-3.2 GlobalProcedures

Unit GlobalProcedures contains all constant, type, and variable declarations, as well as routines that are used in the main program and in several of the supporting procedures. The routines are

1. InitializeVariables--initializes pointer and Boolean variables

2. PrintText--prints a string at a specified location on the screen.

3. ClearWindow--erases a part of the screen.

4. PrintScreen--prints the current screen image.

5. Buzzer--sounds a warning buzzer.

6. MainMenu--prints the main menu choices in the center of the screen.

7. String_to_Value--converts a string containing numbers, multipliers, and units to a numeric value.

8. EXIST--determines if a specified file exists.

9. PrintErrorMsg--prints a message in a box at a specified location on the screen. The routine may optionally wait for a user input before returning control to the calling routine.

```
(*****************************************************************************)
(***                                                                       ***)
(***  These constant, types, variables and procedures are ones used        ***)
(***  throughout the program.                                              ***)
(***                                                                       ***)
(*****************************************************************************)

unit Global;

interface
uses
   DOS,
   Printer,
   Crt,
   Graph;

const
   VersionNumber = 3.00;
   TNNearlyZero  = 1e-15;
   TNArraySize   = 2048;

type
   real          = extended;
   TNVectorArray = array [0..TNArraySize] of real;
   TNVectorPtr   = ^TNVectorArray;

var
   time          : TNVectorPtr;        (* time axis information            *)
   ampl          : TNVectorPtr;        (* amplitude of time domain data    *)
   freq          : TNVectorPtr;        (* frequency axis information       *)
   mag           : TNVectorPtr;        (* magnitude of freq domain data    *)
   phase         : TNVectorPtr;        (* phase of frequency domain data   *)

(*****************************************************************************)

const
   MaxInfo     = 15;                 (* Max num of information lines in file *)
   PI          = 3.141592979431152; (* The constant PI, in double precision *)
   blank       = '     ';            (* Blank string for info                *)
   precision   = 12;                 (* Precision of output file             *)
   ENTER       = #13;                (* ASCII code of RETurn key             *)
   StartColumn = 20;                 (* Left edge of menus                   *)
   MinP        : byte    = 9;        (* Min power of 2 to interpolate to     *)
   SerialPort  : byte    = 1;        (* Comm. port for digitizer.            *)
   NoPrinter   : boolean = true;     (* Printer connected to computer?       *)
   SPLINE      : boolean = true;     (* Cubic Spline or Linear Interp.?      *)
   ForeColor   : byte    = LightGreen; (* Foreground color on color monitor *)
   BackColor   : byte    = Blue;     (* Background color on color monitor    *)
   DrawColor   : byte    = 7;        (* Drawing color for color monitors     *)
   DefaultOptions = 'FFT.cfg';       (* File for default options             *)
   DefaultDataDir : string[79] = 'c:';    (* Default data disk/directory     *)

type
   GraphName    = string;                    (* Descriptors for printing   *)
   InfoArray    = array [1..MaxInfo] of string;  (* Desriptive info in file *)
   PlotType     = 1..4;                      (* Type of plot:              *)
                                             (*    1. Linear               *)
                                             (*    2. Log                  *)
                                             (*    3. Log-Linear           *)
                                             (*    4. Linear-Log           *)

var
   NumPoints  : integer;               (* Number of points read.          *)
```

```
  NumFreqs    : integer;              (* Number of distinct frequencies.   *)
  TRANS       : boolean;              (* Data Translated?                  *)
  ORIG        : boolean;              (* Original Data?                    *)
  DONE        : boolean;              (* Exit program?                     *)
  ACCEPT      : boolean;              (* Interpolated data acceptable?     *)
  HeapTop     : ^integer;             (* Marks top of dynamic memory.      *)
  TempXPtr    : TNVectorPtr;          (* Temporary pointer to TNVector.    *)
  TempYPtr    : TNVectorPtr;          (* Temporary pointer to TNVector.    *)
  OutArrayX   : TNVectorPtr;          (* Temporary output ptr to TNVector. *)
  OutArrayY   : TNVectorPtr;          (* Temporary output ptr to TNVector. *)
  MainChoice  : byte;                 (* Main Menu choice                  *)
  Choice      : char;                 (* temporary variable                *)
  MainChar    : char;                 (* Type of plot (T/M/P)?             *)
  GraphicsCard : integer;             (* Type of graphics card installed   *)
  GraphicsMode : integer;             (* Graphics mode to be used          *)
  XMaxTemp    : integer;              (* Used in window definitions        *)
  YMaxTemp    : integer;              (* Used in window definitions        *)
  leftx       : integer;              (* Left edge of window boundary      *)
  rightx      : integer;              (* Right edge of window boundary     *)
  topy        : integer;              (* Top edge of window boundary       *)
  bottomy     : integer;              (* Bottom edge of window boundary    *)
  i           : integer;              (* Counter variable.                 *)
  info        : InfoArray;            (* Array of information stored at    *)
                                      (* end of data files.                *)

(*****************************************************************************)

procedure Initialize_Variables;


(*****************************************************************************)
(***                                                                       ***)
(***  PrintText prints a string of text, starting at text coordinates      ***)
(***  (xcoor,ycoor); where (1,1) is the upper left corner and (80,25) is   ***)
(***  the lower right corner.                                              ***)
(***                                                                       ***)
(*****************************************************************************)

procedure PrintText (  str      : string;       (* Text to be printed       *)
                       xcoor    : byte;         (* column to start printing *)
                       ycoor    : byte          (* row to start printing    *)
                    );


(*****************************************************************************)
(***                                                                       ***)
(***  ClearWindow is to clear a portion of the screen, from text           ***)
(***  coordinates (XMin,YMin) in the upper left, to (XMax,YMax) in the     ***)
(***  lower right.                                                         ***)
(***                                                                       ***)
(*****************************************************************************)

procedure ClearWindow ( XMin : byte;        (* first column to be erased *)
                        XMax : byte;        (* last  column to be erased *)
                        YMin : byte;        (* first row    to be erased *)
                        YMax : byte         (* last  row    to be erased *)
                      );


(*****************************************************************************)
(***                                                                       ***)
(***  Print_Screen procedure prints the current screen by executing the    ***)
(***  keyboard command "Shift-PrintScreen". Note that the DOS program      ***)
```

```
(***  GRAPHICS.COM must be executed before this routine will work; this   ***)
(***  is done in the initialization section.                              ***)
(***                                                                      ***)
(**************************************************************************)

procedure Print_Screen;


(**************************************************************************)
(***                                                                      ***)
(***  Procedure Buzzer sounds a warning buzzer when an incorrect response ***)
(***  is given by the user.                                               ***)
(***                                                                      ***)
(**************************************************************************)

procedure Buzzer;


(**************************************************************************)
(***                                                                      ***)
(***  MainMenu display the main system menu at the center of the screen.  ***)
(***                                                                      ***)
(**************************************************************************)

procedure MainMenu;


(**************************************************************************)
(***                                                                      ***)
(***  String_to_Value converts a string to a numeric value with a         ***)
(***  multiplier and units.                                               ***)
(***                                                                      ***)
(**************************************************************************)

procedure String_to_Value (     st    : string;
                            VAR value : real;
                            VAR units : string
                          );


(**************************************************************************)
(***                                                                      ***)
(***  Function EXIST determines if the specified filename already exists. ***)
(***                                                                      ***)
(**************************************************************************)

function EXIST (filename : string) : boolean;


(**************************************************************************)
(***                                                                      ***)
(***  PrintErrorMsg prints the string s starting at text coordinates      ***)
(***  (x,y). The message is surrounded by an error box. If wait is true,  ***)
(***  then a char response ,resp, may be read before returning control    ***)
(***  to the calling procedure.                                           ***)
(***                                                                      ***)
(**************************************************************************)

procedure PrintErrorMsg (     s    : string;
                              x    : byte;
                              y    : byte;
                              wait : boolean;
                          VAR resp : char
```

```
                        );

(*****************************************************************************)
(*****************************************************************************)

implementation


procedure Initialize_Variables;

   begin   {Initialize_Variables}
      DONE    :=false;
      TRANS   :=false;
      ORIG    :=false;
      ACCEPT  :=false;
      Mark (HeapTop);
      new (time);
      new (ampl);
      new (freq);
      new (mag);
      new (phase);
      new (TempXPtr);
      new (TempYPtr);
      new (OutArrayX);
      new (OutArrayY);
      FillChar (info,SizeOf(info),0);
   end;   {Initialize_Variables}


procedure PrintText (  str      : string;       (* Text to be printed       *)
                       xcoor    : byte;         (* column to start printing *)
                       ycoor    : byte          (* row to start printing    *)
                    );

   begin   {PrintText}
      GotoXY (xcoor,ycoor);
      write (str);
   end;   {PrintText}


procedure ClearWindow ( XMin : byte;        (* first column to be erased *)
                        XMax : byte;        (* last  column to be erased *)
                        YMin : byte;        (* first row    to be erased *)
                        YMax : byte         (* last  row    to be erased *)
                      );

   var
      CurrMinX : byte;
      CurrMinY : byte;
      CurrMaxX : byte;
      CurrMaxY : byte;

   begin   {ClearWindow}
      CurrMinX:=Lo(WindMin);
      CurrMinY:=Hi(WindMin);
      CurrMaxX:=Lo(WindMax);
      CurrMaxY:=Hi(WindMax);
      Window (XMin,YMin,XMax,YMax);
      ClrScr;
      Window (CurrMinX,CurrMinY,CurrMaxX,CurrMaxY);
   end;   {ClearWindow}
```

```
procedure Print_Screen;

   const FormFeed = #12;     (* formfeed for an Epson-compatible printer     *)

   var
      Reg : Registers;       (* variable to read/write to the 8088 registers *)

   begin   {Print_Screen}
      if NOT NoPrinter then begin
         intr ($5,Reg);
         writeln (lst,FormFeed);
      end;   {if}
   end;   {PrintScreen}


procedure Buzzer;

   const
      pitch = 440;                  (* pitch of warning sound               *)
      PauseLength = 250;            (* length of warning sound, in ms       *)
      LongPauseLength = 1000;       (* additional time to display error msg *)

   begin   {Buzzer}
      Sound (pitch);
      Delay (PauseLength);
      NoSound;
      Delay (LongPauseLength);
   end;   {Buzzer}


procedure MainMenu;

   var
      Choice : char;

   begin   {MainMenu}
      if GraphicsCard = HercMono then begin
         TextColor (ForeColor);
         TextBackground (BackColor);
      end;   {if}
      ClrScr;
      Window (StartColumn-1,2,80,25);
      GotoXY (8,1); write ('Main Menu  (',VersionNumber:4:2,')');
      PrintText ('Select Option by typing a number:',1,3);
      PrintText ('1. Digitizer          ',9,6);
      PrintText ('2. Retrieve/Save Data ',9,8);
      PrintText ('3. Signal Processing  ',9,10);
      PrintText ('4. Graph Results      ',9,12);
      PrintText ('5. Advanced Options   ',9,14);
      PrintText ('6. Create Waveform    ',9,16);
      PrintText ('9. Exit to System     ',9,18);
      PrintText ('Your Choice?          ',1,22);
      GotoXY (14,22);
      repeat
         Choice:=ReadKey;
      until (Choice in ['1'..'6','9']);
      MainChoice:=ord(Choice)-ord('0');
      write (MainChoice);
      Window (1,1,80,25);
   end;   {MainMenu}


procedure String_to_Value (     st    : string;
```

APPENDIX A

```
                        VAR value : real;
                        VAR units : string
                       );

const blank = '    ';

var
   j      : integer;
   code   : integer;
   i      : 0..1;
   factor : real;

begin   {String_to_Value}
   if st[1] = '.' then st:='0'+st;
   Val (st,value,code);
   if code <> 0
      then begin
         i:=0;
         case st[code] of
            'p': factor:=1e-12;          (* pico                    *)
            'n': factor:=1e-09;          (* nano                    *)
            'u': factor:=1e-06;          (* micro                   *)
            'm': factor:=1e-03;          (* milli                   *)
            'k': factor:=1e+03;          (* kilo                    *)
            'M': factor:=1e+06;          (* Mega                    *)
            'G': factor:=1e+09;          (* Giga                    *)
            'T': factor:=1e+12;          (* Tera                    *)
            else begin
                 factor:=1.0;
                 units[1]:=st[code];
                 i:=1;
            end;   {else}
         end;   {case}
         Delete (st,code,1);
         Val (st,value,code);
         if code <> 0
            then begin
               for j:=1 to length(st)-code+1 do
                  units [j+i]:=st[j+code-1];
               Delete (st,code,length(st)-code+1);
               Val (st,value,code);
            end   {then}
            else units:=blank;
          value:=value*factor;
       end   {then}
       else units:=blank;
end;   {String_to_Value}


function EXIST (filename : string) : boolean;

   var
      OK   : boolean;      (* temporary variable, equal to exist *)
      Name : text;

   begin   {EXIST}
      if length (filename) > 0
         then begin
            Assign (Name,filename);
            {$I-} Reset (Name); {$I+}
            OK:=(IOresult=0);
            exist:=OK;
            if OK then Close (Name);
```

```
      end   {then}
      else EXIST:=false;
   end;   {EXIST}


procedure PrintErrorMsg (     s    : string;
                              x    : byte;
                              y    : byte;
                              wait : boolean;
                         VAR resp : char
                        );

   procedure DrawBorder (x1,y1,x2,y2 : byte);

      const
         TopLeft     = '';          {char(201)}
         BottomLeft  = '';          {char(200)}
         TopRight    = '';          {char(187)}
         BottomRight = '';          {char(188)}
         Vertical    = '';          {char(186)}
         Horizontal  = '';          {char(205)}

      begin   {DrawBorder}
         GotoXY (x1,y1); write (TopLeft);
         GotoXY (x1,y2); write (BottomLeft);
         for i:=x1+1 to x2-1 do begin
            GotoXY (i,y1); write (Horizontal);
            GotoXY (i,y2); write (Horizontal);
         end;   {for}
         GotoXY (x2,y1); write (TopRight);
         GotoXY (x2,y2); write (BottomRight);
         for i:=y1+1 to y2-1 do begin
            GotoXY (x1,i); write (Vertical);
            GotoXY (x2,i); write (Vertical);
         end;   {for}
      end;   {DrawBorder}

   const
      PauseLength = 1000;

   var
      CurrMinX     : byte;
      CurrMinY     : byte;
      CurrMaxX     : byte;
      CurrMaxY     : byte;
      MaxLength    : byte;
      BorderLength : byte;

   begin   {PrintErrorMsg}
      CurrMinX:=succ(Lo(WindMin));
      CurrMinY:=succ(Hi(WindMin));
      CurrMaxX:=succ(Lo(WindMax));
      CurrMaxY:=succ(Hi(WindMax));
      MaxLength:=CurrMinX+x+length(s)+2;
      if wait then MaxLength:=succ(MaxLength);
      BorderLength:=MaxLength-CurrMinX+2;
      Window (CurrMinX+x-2,CurrMinY+y-1,MaxLength,CurrMinY+y+1);
      TextColor (White);
      TextBackground (Red);
      ClrScr;
      DrawBorder (1,1,length(s)+4,3);
      PrintText (s,3,2);
      Buzzer;
```

```
      if not wait
         then Delay (PauseLength)
         else resp:=ReadKey;
      TextColor (ForeColor);
      TextBackground (BackColor);
      ClrScr;
      Window (CurrMinX,CurrMinY,CurrMaxX,CurrMaxY);
   end;   {PrintErrorMsg}

(*****************************************************************************)

begin   {Initialization}
   Exec ('\Command.com','/c graphics');
end.   {Initialization}
```

A-3.3 SignalProcessing

Unit SignalProcessing performs two functions: it creates the signal processing menu and it provides access to the data analysis functions. It operates as follows:

1. If necessary, interpolate data

2. Display menu

3. User chooses

    a. Option 1: Integrate
    b. Option 2: Differentiate
    c. Option 3: FFT
    d. Option 4: Inverse FFT
    e. Option 5: Transfer Function
    f. Option 6: Remove Mean
    g. Option 9: Finished

4. Repeat until finished

This file also contains all the signal processing routines, each of which is described below.

Fitting--contains two different fitting routines: a linear and a cubic spline fitting routine. The cubic spline routine is used by default, but the user may select to use linear interpolation from the advanced options menu of the main program. Procedure Fitting also determines the number of points to which to interpolate the data; this value is 2^n for some integer n.

Integrate--performs a numerical integration using the trapezoidal rule.

Differentiate--performs a numerical differentiation using the three point formulas.

TransferFunction--adds, subtracts, multiplies, or divides either axis of the time-domain data (time or amplitude) by a constant.

RemoveMean--removes the mean of the time-domain waveform.

APPENDIX A

MyFFT--contains the actual FFT/IFT routines. These routines were developed from the FFT algorithm given in Burden and Faires, *Trigonometric Polynomial Interpolation* (see bibliography, main body of report). The following routines are contained in this procedure:

1. Timer--gets the current time from DOS, returns the time in seconds.

2. Cooley_Tukey_FFT--the FFT routine itself.

3. PolarForm--converts real/imaginary results to magnitude/phase results.

If the user desires it, the procedure applies a window to the waveform before performing the FFT. Temporary variable reassignments are then performed, the Cooley-Tukey FFT routine is called, the results are put in polar form, and the temporary variables are reassigned to the correct main variables.

After each of the routines is finished, control is returned to the SPS menu.

```
unit SignalProcessing;

interface

uses DOS,
     Crt,
     Global;

(*****************************************************************************)
(***                                                                       ***)
(***  Integrate performs numerical integration on the time domain data     ***)
(***  via the trapezoidal rule. Note that this procedure requires          ***)
(***  approximately 2*m additions and 2*m multiplications.                 ***)
(***                                                                       ***)
(*****************************************************************************)

procedure integrate;


(*****************************************************************************)
(***                                                                       ***)
(***  Differentiate performs numerical differentiation via the so-called   ***)
(***  three point formulae. The routine requires approximately 2*m         ***)
(***  additions and 2*m multiplications.                                   ***)
(***                                                                       ***)
(*****************************************************************************)

procedure differentiate;


(*****************************************************************************)
(***                                                                       ***)
(***  TransferFunction scales either the x- or y-axis by a constant. The   ***)
(***  first character read will decide upon which axis to act, and the     ***)
(***  next character read will decide the operation to be performed. Only  ***)
(***  the characters '*', '/', '+', and '-' are accepted for operations.   ***)
(***                                                                       ***)
(*****************************************************************************)

procedure TransferFunction;


(*****************************************************************************)
(*                                                                           *)
(*  RemoveMean calculates the mean of the time domain data and subtracts     *)
(*  that value from the entire array.                                        *)
(*                                                                           *)
(*****************************************************************************)

procedure RemoveMean;


(*****************************************************************************)
(***                                                                       ***)
(***  Transform performs a forward or inverse fast Fourier transform,      ***)
(***  based on the Cooley-Tukey method. Cooley_Tukey_FFT is modified       ***)
(***  from the FFT algorithm of Numerical Analysis, 3rd ed. by Burden &    ***)
(***  Faires, p. 404. The algorithm is based on the Cooley-Tukey FFT       ***)
(***  algorithm.                                                           ***)
(***                                                                       ***)
(*****************************************************************************)

procedure Transform (INVERSE:boolean);

(*****************************************************************************)
```

APPENDIX A

```
(***                                                                    ***)
(***  SPS prints the SPS menu, and gives the user the choice of one of  ***)
(***  several signal processing functins:                               ***)
(***                                                                    ***)
(***     1. Integration of time domain data.                            ***)
(***     2. Differentiation of time domain data.                        ***)
(***     3. Fast Fourier Transform of time domain data.                 ***)
(***     4. Adjust (+,-,*,/) time domain amplitude by a constant.       ***)
(***     5. Multiply time base by a factor.                             ***)
(***     6. Subtract the mean of the time domain waveform.              ***)
(***                                                                    ***)
(***  SPS is called AFTER the data has been fitted, a procedure invoked ***)
(***  the first time one chooses selection 3 from the Main Menu.        ***)
(***                                                                    ***)
(************************************************************************)

procedure SPS;


(************************************************************************)

implementation

(************************************************************************)
(***                                                                    ***)
(***  Fitting accepts for input an array of NumPoints arbitrarily spaced ***)
(***  data points, and returns as output an array of 2^n evenly spaced  ***)
(***  data points.  The method of interpolation used is either linear   ***)
(***  interpolation or natural cubic spline interpolation.              ***)
(***                                                                    ***)
(************************************************************************)

procedure Fitting;

   const
      Pow2 : array [5..12] of integer = (32,64,128,256,512,1024,2048,4096);

   var
      NewNumPoints : integer;     (* Number of points to interpolate to.   *)
      i            : integer;     (* Counter variable.                     *)
      j            : integer;     (* Counter for NewNumPoints.             *)
      X1           : real;        (* First x value.                        *)
      x            : real;        (* Frequencies of fitted data.           *)
      increment    : real;        (* Distance between successive x-coor.   *)

(************************************************************************)
(***                                                                    ***)
(***  Linear performs linear interpolation on the time domain data.     ***)
(***                                                                    ***)
(************************************************************************)

   procedure Linear;

      var
         m            : real;     (* Slope of interpolation line.          *)
         b            : real;     (* Y-intercept of interpolation line.    *)
         denom        : real;     (* Difference between succesive points.  *)
         j            : integer;
      begin   (Linear)
         i:=0;
         X1:=time^[0];
         TempXPtr^[0]:=time^[0];
         TempYPtr^[0]:=ampl^[0];
         increment:=(time^[NumPoints-1]-time^[0])/(NewNumPoints-1);
         for j:=1 to NewNumPoints-1 do begin
            x:=X1+j*increment;
```

```
            while x > time^[i] do i:=succ(i);
            (*** Calculate slope and y-intercept ***)
            denom:=time^[i]-time^[i-1];
            m:=(ampl^[i]-ampl^[i-1])/denom;
            b:=(ampl^[i-1]*time^[i]-ampl^[i]*time^[i-1])/denom;
            TempXPtr^[j]:=x;
            TempYPtr^[j]:=m*x+b;
         end;   (for)
      end;   (Linear)

(*****************************************************************************)
(***                                                                       ***)
(***  CubicSpline performs a Cubic Spline interpolation on the time        ***)
(***  domain data. The algorithm used is a modification of the algorithm   ***)
(***  3.4 of Numerical Analysis, 3rd ed. by Burden & Faires, p. 122.       ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure CubicSpline;

      var
         temp          : real;        (* A temporary real variable.          *)
         b             : TNVectorPtr; (* First spline coefficient.           *)
         c             : TNVectorPtr; (* Second spline coefficient.          *)
         d             : TNVectorPtr; (* Third spline coefficient.           *)
         h             : TNVectorPtr; (* Dist. between successive data pts.  *)
         i             : integer;
         j             : integer;

      begin   (CubicSpline)
         new (b); new (c); new (d); new (h);
         (*** Calculate spline coefficients ***)
         for i:=0 to NumPoints-2 do h^[i]:=time^[i+1]-time^[i];
         b^[0]:=1;
         OutArrayX^[0]:=0;
         OutArrayY^[0]:=0;
         for i:=1 to NumPoints-2 do begin
            temp:=3*(ampl^[i+1]*h^[i-1]-ampl^[i]*(time^[i+1]-time^[i-1])
                  +ampl^[i-1]*h^[i])/(h^[i-1]*h^[i]);
            b^[i]:=2*(time^[i+1]-time^[i-1])-h^[i-1]*OutArrayX^[i-1];
            OutArrayX^[i]:=h^[i]/b^[i];
            OutArrayY^[i]:=(temp-h^[i-1]*OutArrayY^[i-1])/b^[i];
         end;   (for)
         c^[NumPoints-1]:=0;
         for i:=NumPoints-2 downto 0 do begin
            c^[i]:=OutArrayY^[i]-OutArrayX^[i]*c^[i+1];
            b^[i]:=(ampl^[i+1]-ampl^[i])/h^[i]-h^[i]*(c^[i+1]+2*c^[i])/3;
            d^[i]:=(c^[i+1]-c^[i])/(3*h^[i]);
         end;   (for)

         (*** Create NewNumPoints evenly spaced points ***)
         X1:=time^[0];
         increment:=(time^[NumPoints-1]-time^[0])/(NewNumPoints-1);
         i:=0;
         TempXPtr^[0]:=time^[0];
         TempYPtr^[0]:=ampl^[0];
         for j:=1 to NewNumPoints-1 do begin
            x:=X1+j*increment;
            while x > time^[i] do i:=succ(i);
            temp:=x-time^[i-1];
            TempXPtr^[j]:=x;
            TempYPtr^[j]:=ampl^[i-1]+b^[i-1]*temp+c^[i-1]*sqr(temp)
                          +d^[i-1]*sqr(temp)*temp;
         end;   (for)
         dispose (b); dispose (c); dispose (d); dispose (h);
      end;   (CubicSpline)
```

APPENDIX A

```
   begin   (Fitting)
      (*** Calculate number of points to interpolate ***)
      (*** to. Must be a power of two.                ***)
      i:=MinP;
      while NumPoints > Pow2[i] do i:=succ(i);
      NumFreqs:=Pow2[i-1];
      NewNumPoints:=Pow2[i];
      if SPLINE
         then CubicSpline
         else Linear;
      NumPoints:=NewNumPoints;
      ACCEPT:=true;
      time^:=TempXPtr^;
      ampl^:=TempYPtr^;
   end;   (Fitting)

(*****************************************************************************)

procedure integrate;

   var
      i   : integer;          (* Counter.                                    *)
      h   : real;             (* Distance between successive points.         *)

   begin   (integrate)
      h:=(time^[NumPoints-1]-time^[0])/(NumPoints);
      TempXPtr^[0]:=0;
      for i:=1 to NumPoints-1 do
         TempXPtr^[i]:=TempXPtr^[i-1]+0.5*h*(ampl^[i-1]+ampl^[i]);
      for i:=0 to NumPoints-1 do
         ampl^[i]:=TempXPtr^[i];
   end;   (integrate)

(*****************************************************************************)

procedure differentiate;

   var
      Two_m : integer;        (* Two_m = 2*m-1 = Num points in input array.  *)
      h     : real;           (* Distance between successive points.         *)
      i     : integer;        (* Counter.                                    *)

   begin   (differentiate)
      Two_m:=NumPoints-1;
      h:=(time^[Two_m]-time^[0])/(Two_m+1);
      TempXPtr^[0]:=0.5*(-3*ampl^[0]+4*ampl^[1]-ampl^[2])/h;
      TempXPtr^[Two_m]:=0.5*(3*ampl^[Two_m]-4*ampl^[Two_m-1]+ampl^[Two_m-2])/h;
      for i:=1 to Two_m-1 do
         TempXPtr^[i]:=0.5*(ampl^[i+1]-ampl^[i-1])/h;
      for i:=1 to Two_m do
         ampl^[i]:=TempXPtr^[i];
   end;   (differentiate)

(*****************************************************************************)

procedure TransferFunction;

   var
      i         : integer;                (* counter variable           *)
      s         : string;
      transfer  : real;                   (* scaling factor             *)
      operation : char;                   (* operation to be performed  *)
      X_AXIS    : boolean;                (* apply to x or y axis?      *)

   begin   (TransferFunction)
```

```
      PrintText ('Apply to x or y axis? ',1,21);
      repeat
         operation:=ReadKey;
      until (UpCase(operation) in ['X','Y']);
      if UpCase(operation) = 'X'
         then X_AXIS:=true
         else X_AXIS:=false;
      PrintText ('Factor:        ',1,23);
      GotoXY (10,23);
      repeat
         operation:=ReadKey;
      until (operation in ['*','/','+','-']);
      write (operation);
      readln (s);
      String_To_Value (s,transfer,s);
      if X_AXIS
         then case operation of
            '*': for i:=0 to NumPoints-1 do time^[i]:=time^[i]*transfer;
            '/': for i:=0 to NumPoints-1 do time^[i]:=time^[i]/transfer;
            '+': for i:=0 to NumPoints-1 do time^[i]:=time^[i]+transfer;
            '-': for i:=0 to NumPoints-1 do time^[i]:=time^[i]-transfer;
         end   {case}
         else case operation of
            '*': for i:=0 to NumPoints-1 do ampl^[i]:=ampl^[i]*transfer;
            '/': for i:=0 to NumPoints-1 do ampl^[i]:=ampl^[i]/transfer;
            '+': for i:=0 to NumPoints-1 do ampl^[i]:=ampl^[i]+transfer;
            '-': for i:=0 to NumPoints-1 do ampl^[i]:=ampl^[i]-transfer;
         end;   {case}
   end;   {TransferFunction}

(*****************************************************************************)
(*                                                                           *)
(*  RemoveMean calculates the mean of the time domain data and subtracts     *)
(*  that value from the entire array.                                        *)
(*                                                                           *)
(*****************************************************************************)

procedure RemoveMean;

   var
      i   : integer;              (* Counter variable.                      *)
      sum : real;                 (* The mean of the array.                 *)

   begin   {RemoveMean}
      sum:=0;
      for i:=0 to NumPoints-1 do
         sum:=sum+ampl^[i];
      sum:=sum/(NumPoints);
      for i:=0 to NumPoints-1 do
         ampl^[i]:=ampl^[i]-sum;
   end;   {RemoveMean}

(*****************************************************************************)

procedure Transform (INVERSE:boolean);

   const
      MaxPower = 12;

   type
      polar    = record
                    Mag : real;
                    Ph  : real;
                 end;   {record}

   var
```

```
      beta       : real;                    (* beta = t,max - t,min          *)
      delta      : real;                    (* delta = 1/beta                *)
      i          : integer;                 (* counter variable              *)
      j          : integer;                 (* counter variable              *)
      ans        : char;                    (* Apply window?                 *)
      n          : integer;                 (* n = 2*NumFreqs-1              *)
      p          : 1..MaxPower;             (* Power of two of FFT           *)
      sign       : integer;                 (* Sign of real part             *)
      magnitude  : real;                    (* Magnitude of real part        *)
      StartTime  : real;                    (* Time at beginning of FFT      *)
      EndTime    : real;                    (* Time when finished FFT        *)
      delta_t    : real;                    (* Time to perform FFT           *)

   function timer: real;

      var
         hour,min,sec,ms : word;

      begin
         GetTime (hour,min,sec,ms);
         timer:=hour*3600 + min*60 + sec + ms/100;
      end;

(*****************************************************************************)
(***                                                                       ***)
(***  Cooley_Tukey_FFT is modified from the FFT algorithm of Numerical     ***)
(***  Analysis, 3rd ed. by Burden & Faires, p. 404. The algorithm is       ***)
(***  based on the Cooley-Tukey FFT algorithm.                             ***)
(***                                                                       ***)
(*****************************************************************************)


   procedure Cooley_Tukey_FFT;

      type
         binary      = 0..1;
         BinaryArray = array [0..MaxPower] of binary;

      var
         big_M    : integer;      (*                                        *)
         big_K    : integer;      (*                                        *)
         q        : integer;      (*                                        *)
         L        : integer;      (*                                        *)
         k        : BinaryArray;  (*                                        *)
         K2       : integer;      (* Subscript after bit reversal           *)
         eta      : polar;        (*                                        *)
         temp     : polar;        (* Used to switch two values.             *)
         i        : integer;
         j        : integer;

      begin   {Cooley_Tukey_FFT}
         OutArrayX^[0]:=1;
         OutArrayY^[0]:=0;
         for i:=1 to NumFreqs do begin
            OutArrayX^[i]:=cos(i*PI/NumFreqs);
            OutArrayY^[i]:=sin(i*PI/NumFreqs);
            OutArrayX^[i+NumFreqs]:=-OutArrayX^[i];
            OutArrayY^[i+NumFreqs]:=-OutArrayY^[i];
         end;   {for}
         big_K:=0;
         big_M:=NumFreqs;
         q:=p;
         for L:=1 to succ(p) do begin
            while big_K < n do begin
               for j:=1 to big_M do begin
                  for i:=0 to p do
```

```
                  k[i]:=binary((big_K AND (1 shl i)) shr i);
               K2:=0;
               for i:=p downto q do
                  K2:=K2+(k[i] shl (p-i+q));
               eta.Mag:=TempXPtr^[big_K+big_M]*OutArrayX^[K2]
                       -TempYPtr^[big_K+big_M]*OutArrayY^[K2];
               eta.Ph :=TempXPtr^[big_K+big_M]*OutArrayY^[K2]
                       +TempYPtr^[big_K+big_M]*OutArrayX^[K2];
               TempXPtr^[big_K+big_M]:=TempXPtr^[big_K]-eta.Mag;
               TempYPtr^[big_K+big_M]:=TempYPtr^[big_K]-eta.Ph;
               TempXPtr^[big_K]:=TempXPtr^[big_K]+eta.Mag;
               TempYPtr^[big_K]:=TempYPtr^[big_K]+eta.Ph;
               big_K:=succ(big_K);
            end;   (for)
            big_K:=big_K+big_M;
         end;   (while)
         big_K:=0;
         big_M:=big_M div 2;
         q:=pred(q);
      end;   (for)
      while big_K < n do begin
         for i:=0 to p do
            k[i]:=binary((big_K AND (1 shl i)) shr i);
         j:=0;
         for i:=0 to p do
            j:=j+(k[i] shl (p-i));
         if j>big_K then begin
            temp.Mag:=TempXPtr^[j];
            temp.Ph :=TempYPtr^[j];
            TempXPtr^[j].=TempXPtr^[big_K];
            TempYPtr^[j]:=TempYPtr^[big_k];
            TempXPtr^[big_K]:=temp.Mag;
            TempYPtr^[big_K]:=temp.Ph;
         end;   (if)
         big_K:=succ(big_K);
      end;   (while)
   end;   (Cooley_Tukey_FFT)

(*************************************************************************)
(***                                                                   ***)
(***  Results of previous procedures in Transform are given as real and ***)
(***  imaginary parts. PolarForm converts those values to magnitude and ***)
(***  phase (radians) components.                                      ***)
(***                                                                   ***)
(***  Arg = the counterclockwise angle between the positive half of the ***)
(***  real axis, and the ray from the origin to the complex number.    ***)
(***                                                                   ***)
(*************************************************************************)

   procedure PolarForm;

      const
         epsilon1 = 1e-06;      (* A tolerance. Anything  <= epsilon1 is 0. *)
         epsilon2 = 1e-02;      (* A tolerance. Any angle <= epsilon2 is 0. *)

      var
         i      : integer;      (* A counter variable.                      *)
         temp   : polar;        (* A temporary variable.                    *)
         konst1 : real;         (* Correction factor for phase angle.       *)
         konst2 : real;         (* Magnitude scale factor for forward FFT.  *)
         Max    : real;         (* Maximum magnitude of frequency data.     *)

      function Arg (v:polar) : real;

         begin   (Arg)
            if v.Mag>=0
```

```
            then konst1:=0
         else if v.Ph>=0
            then konst1:=PI
            else konst1:=-PI;
         if abs(v.Mag) <= epsilon1
            then if v.Ph>=0
               then Arg:=+PI/2
               else Arg:=-PI/2
            else Arg:=arctan(v.Ph/v.Mag)+konst1;
      end;   {Arg}

   begin   {PolarForm}
      konst2:=2*NumFreqs*delta;
      for i:=0 to n do begin
         temp.Mag:=TempXPtr^[i];
         temp.Ph :=TempYPtr^[i];
         TempXPtr^[i]:=sqrt(sqr(temp.Mag)+sqr(temp.Ph))/konst2;
         TempYPtr^[i]:=Arg (temp);
      end;   {for}
      (*** Eliminate "unreal" phase angles. ***)
      Max:=TempXPtr^[0];
      for i:=1 to n do
         if (Max < TempXPtr^[i]) then Max:=TempXPtr^[i];
      for i:=0 to n do
         if (TempXPtr^[i] < Max*epsilon2) then TempYPtr^[i]:=0;
   end;   {PolarForm}

(*****************************************************************************)

   begin   {Transform}
      p:=round(ln(NumFreqs)/ln(2));
      n:=2*NumFreqs-1;      (* = NumPoints-1 *)

      if NOT INVERSE
         then begin
            (*** Should a window be applied to the time domain data? ***)
            PrintText ('Apply window [H/R/N]? ',1,22);
            ClrEOL;
         end   {then}
         else begin
            PrintText ('Inverse FFT in progress. Please wait ...',1,22);
            ClrEOL;
         end;   {else}

      (*** Make variable assignments while user reads prompt. ***)
      if INVERSE
         then begin
            beta:=n/(freq^[n]-freq^[0]);
            (*** Reorder data for inverse FFT ***)
            for i:=0 to n do begin
               TempXPtr^[i]:=mag^[i]*cos(phase^[i]);
               TempYPtr^[i]:=mag^[i]*sin(phase^[i]);
            end;   {for}
            delta:=beta/(n+1);
         end   {then}
         else begin
            for i:=0 to n do begin
               TempXPtr^[i]:=ampl^[i];
               TempYPtr^[i]:=0;
            end;   {for}
            delta:=1/(time^[n]-time^[0]);
         end;   {else}

      if NOT INVERSE then begin
         GotoXY (23,22); ans:=ReadKey;
         if (UpCase(ans)='H') then
```

```
            for i:=Ø to n do                (*** Apply Hanning window ***)
               TempXPtr^[i]:=TempXPtr^[i]*(Ø.5*(1-cos(PI*i/NumFreqs)))
         else if (UpCase(ans)='R') then
            for i:=Ø to n do                (*** Apply rear window ***)
               TempXPtr^[i]:=TempXPtr^[i]*(Ø.5*(1-cos((Ø.5*PI*i/NumFreqs)+PI)));
         if ans <> ENTER
            then write (ans)
            else write ('N');
         PrintText ('FFT in progress. Please wait ...',1,22);
         ClrEOL;
      end;   {if}

      (*** Perform FFT or IFT on data. ***)
      StartTime:=timer;
      Cooley_Tukey_FFT;
      EndTime:=timer;
      delta_t:=EndTime-StartTime;
      GotoXY (1,2Ø);
      write ('FFT performed in ',delta_t:6:1,' seconds.');

      (*** Prepare output arrays. ***)
      if INVERSE
         then begin
            for i:=Ø to n do begin
               sign:=round(TempXPtr^[n-i]/abs(TempXPtr^[n-i]));
               magnitude:=sqrt(sqr(TempXPtr^[n-i])+sqr(TempYPtr^[n-i]));
               time^[i]:=i*delta;
               ampl^[i]:=sign*magnitude/beta;
            end;   {for}
         end   {then}
         else begin
            (*** Convert real and imaginary parts ***)
            (*** to magnitude and phase parts.    ***)
            PolarForm;
            for i:=Ø to n do begin
               freq^[i] :=i*delta;
               mag^[i]  :=TempXPtr^[i];
               phase^[i]:=TempYPtr^[i];
            end;   {for}
       end;   {else}
   end;   {Transform}

(*****************************************************************************)
(***                                                                       ***)
(***  SPS prints the SPS menu, and gives the user the choice of one of     ***)
(***  several signal processing functins:                                  ***)
(***                                                                       ***)
(***     1. Integration of time domain data.                               ***)
(***     2. Differentiation of time domain data.                           ***)
(***     3. Fast Fourier Transform of time domain data.                    ***)
(***     4. Adjust (+,-,*,/) time domain amplitude by a constant.          ***)
(***     5. Multiply time base by a factor.                                ***)
(***     6. Subtract the mean of the time domain waveform.                 ***)
(***                                                                       ***)
(***  SPS is called AFTER the data has been fitted, a procedure invoked    ***)
(***  the first time one chooses selection 3 from the Main Menu.           ***)
(***                                                                       ***)
(*****************************************************************************)

procedure SPS;

   procedure SPSMenu;

      var
         Choice : char;
```

APPENDIX A

```
      begin   {SPSMenu}
         ClrScr;
         PrintText ('Signal Processing Menu',6,1);
         PrintText ('Select Option by typing a number: ',1,3);
         PrintText ('1. Integrate          ',9,6);
         PrintText ('2. Differentiate      ',9,8);
         PrintText ('3. FFT                ',9,10);
         PrintText ('4. Inverse FFT        ',9,12);
         PrintText ('5. Transfer Function  ',9,14);
         PrintText ('6. Remove DC Level    ',9,16);
         PrintText ('9. Exit to Main Menu  ',9,18);
         PrintText ('Your Choice?          ',1,22);
         GotoXY (14,22);
         repeat
            Choice:=ReadKey;
         until Choice in ['1'..'6','9'];
         MainChoice:=ord(Choice)-ord('0');
         write (MainChoice);
      end;   {SPSMenu}

   begin   {SPS}
      Window (StartColumn-1,2,80,25);
      DONE:=false;
      if NOT ACCEPT then Fitting;
      repeat
         SPSMenu;
         case MainChoice of
            1: Integrate;
            2: Differentiate;
            3: begin   {Transform}
                  Transform (false);
                  TRANS:=true;
                  ORIG:=false;
               end;   {Transform}
            4: if TRANS then begin
                  Transform (true);
                  ORIG:=true;
               end;   {Inverse FFT}
            5: TransferFunction;
            6: RemoveMean;
            9: DONE:=true;
         end;   {case}
      until DONE;
      DONE:=false;
      Window (1,1,80,25);
      ClrScr;
   end;   {SPS}

(*****************************************************************************)

begin   {Initialization}
end.   {Initialization}
```

A-3.4 AdvancedOpts

Unit AdvancedOpts contains two routines:

1. AdvancedOptions and

2. Read_DefaultOptions.

Procedure AdvancedOptions allows the user to change the type of interpolation used, the background and foreground colors used on color monitors, the port to which the digitizer is connected, whether a printer is installed, and the default drive/directory for data. The user may also save these options to a disk file called "FFT.cfg" on the current drive/directory.

Procedure Read_DefaultOptions reads the options saved to disk from the file "FFT.cfg" if it exists. This file is only read when program FFT is first started.

APPENDIX A

```
unit AdvancedOpts;

interface

uses
   Crt,
   Graph,
   Global;


(*****************************************************************************)
(***                                                                       ***)
(***  AdvancedOptions allows several options to be changed.                ***)
(***                                                                       ***)
(*****************************************************************************)

procedure AdvancedOptions;


(*****************************************************************************)
(***                                                                       ***)
(***  Read_Default_Options reads the default options file that was saved   ***)
(***  previously by the user under the advanced options menu.              ***)
(***                                                                       ***)
(*****************************************************************************)

procedure Read_DefaultOptions;


(*****************************************************************************)
(*****************************************************************************)

implementation

procedure AdvancedOptions;

   const
      MaxP   = 12;
      Pow2   : array [5..12] of integer
             = (32,64,128,256,512,1024,2048,4096);

   var
      choice : byte;
      int    : char;
      DONE   : boolean;
      NewNum : integer;
      temp   : string;
      Name   : text;
      error  : integer;
      ch     : char;
      Drive  : byte;
      CurrentDir : string;

   procedure AdvancedOptionsMenu;

      begin   {AdvancedOptionsMenu}
         ClrScr;
         Window (StartColumn-1,2,80,25);
         PrintText ('Advanced Options',9,1);
         PrintText ('Select Option by typing a number: ',1,3);
         PrintText ('1. Linear/Spline Interpolation:   ',6,6);
         PrintText ('2. Digitizer Port [0/1/2]:        ',6,8);
         PrintText ('3. Printer installed (y/n) ?      ',6,10);
```

```
      PrintText ('4. Colors:                          ',6,12);
      PrintText ('5. Data Disk/Directory:             ',6,14);
      PrintText ('6. Save Options                     ',6,16);
      PrintText ('9. Exit to Main Menu                ',6,18);
      PrintText ('Your Choice?                        ',1,22);
      GotoXY (40,6);
      if SPLINE
         then write ('S')
         else write ('L');
      GotoXY (40,8);  write (SerialPort);
      GotoXY (40,10);
      if NoPrinter
         then write ('N')
         else write ('Y');
      GotoXY (33,12);
      write (ForeColor:2,'/',BackColor:2,'/',DrawColor:2);
      GotoXY (30,14); write (DefaultDataDir:11);
      DONE:=false;
   end;   {AdvancedOptionsMenu}

begin   {AdvancedOptions}
   AdvancedOptionsMenu;
   repeat
      GotoXY (14,22);
      repeat
         ch:=ReadKey;
      until ch in ['1'..'6','9'];
      choice:=ord(ch)-ord('0');
      write (ch);
      case choice of
         1: begin
               GotoXY (40,6);
               if SPLINE
                  then begin
                     SPLINE:=false;
                     write ('L');
                  end   {then}
                  else begin
                     SPLINE:=true;
                     write ('S');
                  end;   {else}
               ACCEPT:=false;
            end;
         2: begin
               GotoXY (40,8);
               repeat
                  ch:=ReadKey;
               until ch in ['0'..'2'];
               SerialPort:=ord(ch)-ord('0');
               write (SerialPort);
            end;
         3: begin
               GotoXY (40,10);
               if NoPrinter
                  then begin
                     NoPrinter:=false;
                     write ('Y');
                  end   {then}
                  else begin
                     NoPrinter:=true;
                     write ('N');
                  end;   {else}
            end;
```

```
4: begin
      GotoXY (30,12); ClrEOL;
      GotoXY (1,22);
      write ('Foreground color [',ForeColor,'] ? ');
      ClrEol;
      readln (temp);
      Val (temp,NewNum,error);
      if error = 0 then ForeColor:=NewNum mod 16;
      GotoXY (1,22);
      write ('Background color [',BackColor,'] ? ');
      ClrEol;
      readln (temp);
      Val (temp,NewNum,error);
      if error = 0 then BackColor:=NewNum mod 16;
      GotoXY (1,22);
      write ('Drawing color [',DrawColor,'] ? ');
      ClrEol;
      readln (temp);
      Val (temp,NewNum,error);
      if error = 0 then DrawColor:=NewNum mod 16;
      if GraphicsCard <> HercMono then begin
         TextColor (ForeColor);
         TextBackground (BackColor);
      end;   {if}
      Window (1,1,80,25);
      AdvancedOptionsMenu;
   end;
5: begin
      PrintText ('New data disk/directory? ',1,22);
      readln (temp);
      if length (temp) > 0 then begin
         Drive:=0;
         GetDir (Drive,CurrentDir);
         {$I-} ChDir (temp); {$I+}
         if IOResult = 0
            then begin
               DefaultDataDir:=temp;
               ChDir (CurrentDir);
               GotoXY (30,14);
               write (DefaultDataDir:11); ClrEOL;
            end   {then}
            else begin
               PrintErrorMsg
                       ('Invalid disk/directory! ',1,19,false,ch);
            end;   {else}
      end;   {if}
      PrintText ('Your Choice? ',1,22); ClrEOL;
   end;
6: begin
      PrintText ('Saving Options. Please wait ...',1,22);
      Assign (Name,DefaultOptions);
      Rewrite (Name);
      if SPLINE
         then writeln (Name,'SPLINE Interpolation')
         else writeln (Name,'LINEAR Interpolation');
      writeln (Name,'Digitizer at COM',SerialPort,':');
      if NoPrinter
         then writeln (Name,'NO Printer installed')
         else writeln (Name,'Printer is installed');
      writeln (Name,'Foreground Color is ',ForeColor:2);
      writeln (Name,'Background Color is ',BackColor:2);
      writeln (Name,'Drawing Color is ',DrawColor:2);
      writeln (Name,'Default data directory is ',DefaultDataDir);
```

```
                    Close (Name);
                    PrintText ('Your Choice? ',1,22); ClrEOL;
                 end;
              9: begin
                    DONE:=true;
                    Window (1,1,80,25);
                    ClrScr;
                 end;
           end;   {case}
           PrintText (' ',14,22);
        until DONE;
     end;   {AdvancedOptions}


procedure Read_DefaultOptions;

     var
        Name  : text;
        temp  : string;
        i,j   : byte;
        error : integer;
        num   : string [2];

     begin   {Read_DefaultOptions}
        if EXIST (DefaultOptions) then begin
           Assign (Name,DefaultOptions);
           Reset (Name);
           readln (Name,temp);
           i:=Pos ('SPLINE',temp);
           if i >= 1
              then SPLINE:=true
              else SPLINE:=false;
           readln (Name,temp);
           i:=Pos ('COM',temp);
           SerialPort:=ord(temp[i+3])-ord('0');
           if NOT (SerialPort in [0..2]) then SerialPort:=0;
           readln (Name,temp);
           if temp[1] in ['P','p']
              then NoPrinter:=false
              else NoPrinter:=true;
           readln (Name,temp);
           i:=Pos ('is ',temp);
           num:=copy (temp,i+3,2);
           Val (num,i,error);
           if error = 0 then ForeColor:=i;
           readln (Name,temp);
           i:=Pos ('is ',temp);
           num:=copy (temp,i+3,2);
           Val (num,i,error);
           if error = 0 then BackColor:=i;
           readln (Name,temp);
           i:=Pos ('is ',temp);
           num:=copy (temp,i+3,2);
           Val (num,i,error);
           if error = 0 then DrawColor:=i;
           readln (Name,temp);
           i:=Pos ('is ',temp);
           if i > 0 then begin
              for j:=i+3 to length(temp) do
                 DefaultDataDir[j-i-2]:=temp[j];
              DefaultDataDir[0]:=char (length(temp)-i-2);
           end;   {if}
        end;   {if}
```

```
    end;   {Read_DefaultOptions}


(*****************************************************************************)
(*****************************************************************************)

begin   {Initialization}
end.   {Unit AdvancedOpts}
```

A-3.5 FileSystem

Unit FileSystem performs all the file operations of this program. FileIO first sets up a menu, then saves, retrieves, or deletes a file from disk. In addition, a file may be imported from the Device Damage Testing (DDT) programs written by James Loftus of Harry Diamond Laboratories, a disk directory may be listed, or the information lines of a data set may be displayed. The following procedures and functions are contained in this file:

1. SaveFile--saves data to disk.

2. RetrieveFile--retrieves data from a disk file.

3. DeleteFile--deletes a disk file.

4. ImportDDT--imports a DDT file.

5. Directory--gives a directory listing.

6. ReadInfoLines--displays the information lines of a data set.

APPENDIX A

```
(*****************************************************************************)
(***                                                                       ***)
(***  FileIO saves, retrieves, or deletes files from disk. Data files      ***)
(***  will be saved as follows:                                            ***)
(***        1. Number of elements.                                         ***)
(***           if ORIG then this is NumPoints                              ***)
(***           if TRANS then this is 2*m.                                  ***)
(***        2. x, y                    (the time domain data point)        ***)
(***     if applicable, the following is then saved:                       ***)
(***        3. freq., mag., phase      (the frequency domain points)       ***)
(***        4. From Ø up to MaxInfo number of information lines.           ***)
(***                                                                       ***)
(***  Additionally, files may be "imported" from the Device Damage         ***)
(***  Testing (DDT) programs. When imported, the DDT file is translated    ***)
(***  to the format listed above. The operator is also given the option    ***)
(***  of scaling the integer values of the DDT files to "true" values.     ***)
(***                                                                       ***)
(***  A directory may also be shown by giving a file mask. The file mask   ***)
(***  is of the form [d:][path][filename][.ext]. Wildcards may be used.    ***)
(***                                                                       ***)
(*****************************************************************************)

unit FileSystem;

interface
uses
   DOS,
   Crt,
   Global;


procedure FileIO;


(*****************************************************************************)

implementation

var
   Response,Choice : shortint;      (* FileIO menu choices              *)
   counter         : integer;       (* counter variables                *)
   OKSave          : boolean;       (* overwrite old file?              *)
   IODONE          : boolean;       (* exit FileIO?                     *)
   Name            : text;          (* file input/output stream         *)
   RespCh          : char;

(*****************************************************************************)
(***                                                                       ***)
(***  SaveFile tests if the specified filename already exists. If so, the  ***)
(***  user is asked if the old file should be overwritten. If so, or if    ***)
(***  the old file does not exist, then the data is saved as filename and  ***)
(***  the user is returned to the Main Menu, otherwise the user is         ***)
(***  returned to the FileIO Menu.                                         ***)
(***                                                                       ***)
(*****************************************************************************)

procedure SaveFile;

   var
      i : integer;
      filename : string;

   begin  {SaveFile}
      OKSave:=false;
      IODONE:=false;
      FillChar (filename,SizeOf(filename),Ø);
```

```
      filename[0]:=char(0);
      PrintText ('Save to filename: ',1,22);
      ReadLn (filename);
      if filename[2] <> ':' then if filename[1] <> '\' then
         filename:=DefaultDataDir+'\'+filename;
      (*** Does file exist? If so, is it safe to overwrite it? ***)
      if NOT EXIST (filename)
         then begin
            if length (filename) > 0
               then OKSave:=true
               else OKSave:=false;
         end   (then)
         else begin
            PrintErrorMsg ('File already exists: overwrite (y/n)? ',
                                                  1,19,true,RespCh);
            if (UpCase(char(RespCh))='Y')
               then OKSave:=true
               else OKSave:=false;
         end;   (else)
      (*** If it is safe to write to the file, then save the data. ***)
      if OKSave
         then begin
            {$I-}    (* Turn off error checking. *)
            PrintText ('Saving. Please wait...          ',1,20);
            Assign (Name,filename);
            Rewrite (Name);
            if TRANS
               then writeln (Name,2*NumFreqs)
               else writeln (Name,NumPoints);
            if ORIG
               then for i:=0 to NumPoints-1 do
                  writeln (Name,time^[i]:precision,'   ',
                                ampl^[i]:precision)
               else for i:=0 to 2*NumFreqs-1 do
                  writeln (Name,time^[i]:precision,'   ',
                                ampl^[i]:precision,'   ',
                                freq^[i]:precision,'   ',
                                mag^ [i]:precision,'   ',
                                phase^[i]:precision);
            for i:=1 to MaxInfo do
               writeln (Name,info[i]);
            Close (Name);
            PrintText ('                                  ',1,20);
            PrintText ('                                  ',1,22);
            {$I+}   (* Turn error checking on. *)
            if (IOResult <> 0) then begin
               PrintErrorMsg ('Error on disk. File not saved! ',
                                          1,22,false,RespCh);
            end;   (if)
         end   (then)
         else begin
            PrintText ('Your Choice?                    ',1,22);
         end;   (else)
      end;   (SaveFile)

(*****************************************************************************)
(***                                                                       ***)
(***  RetrieveFile first calls EXIST. If the file exists, then the file    ***)
(***  is read and the user is returned to the Main Menu; otherwise the     ***)
(***  user is given an error message and returned to the FileIO Menu.      ***)
(***                                                                       ***)
(*****************************************************************************)


procedure RetrieveFile;
```

APPENDIX A

```
var
   j : integer;          (* temporary counter variable                *)
   x : real;             (* dummy variable                            *)
   y : real;             (* dummy variable                            *)
   i : integer;
   filename : string;

begin  (RetrieveFile)
   PrintText ('                                    ',1,2Ø);
   PrintText ('Read filename: ',1,22);
   ReadLn (filename);
   if filename[2] <> ':' then if filename[1] <> '\' then
      filename:=DefaultDataDir+'\'+filename;
   (*** Does file exist? if not, ERROR! If so, read the file. ***)
   if NOT EXIST (filename)
      then begin
         PrintErrorMsg ('File does not exist!',1,19,false,RespCh);
         PrintText ('Your Choice? ',1,22);
      end   (then)
      else begin
         PrintText ('Reading. Please wait ...  ',1,2Ø);
         Assign (Name,filename);
         Reset (Name);
         TRANS:=false;
         ORIG:=false;
         ACCEPT:=false;
         Readln (Name,NumPoints);
         read (Name,x,y);
         if NOT EOLN (Name)
            then begin
               readln (Name,freq^[Ø],mag^[Ø],phase^[Ø]);
               TRANS:=true;
            end   (then)
            else begin
               ORIG:=true;
               freq^[Ø]:=Ø;
               mag^[Ø]:=Ø;
               phase^[Ø]:=Ø;
            end;   (else)
         time^[Ø]:=x;
         ampl^[Ø]:=y;
         if TRANS
            then for i:=1 to NumPoints-1 do begin
               Readln (Name,time^[i],
                           ampl^[i],
                           freq^[i],
                           mag^[i],
                           phase^[i]);
            end   (then for)
            else for i:=1 to NumPoints-1 do begin
               Readln (Name,time^[i],
                           ampl^[i]);
               freq^[i] :=Ø;
               mag^[i]  :=Ø;
               phase^[i]:=Ø;
            end;   (else for)
         if TRANS
            then begin
               NumFreqs:=NumPoints div 2;
               ACCEPT:=true;
            end   (then)
            else begin
               ORIG:=true;
            end;   (else)
         i:=1;
         while (NOT EOF (Name)) AND (i <= MaxInfo) do begin
```

```
            Readln (Name,info[i]);
            i:=succ(i);
         end;   (while)
         for j:=i+1 to MaxInfo do
            info[i]:=blank;
         Close (Name);
      end;   (else)
   end;   (RetrieveFile)

(*****************************************************************************)
(***                                                                       ***)
(***  DeleteFile deletes a file, if present, from disk, after asking the   ***)
(***  user to verify that he wants to delete it.                           ***)
(***                                                                       ***)
(*****************************************************************************)

procedure DeleteFile;

   var
      filename : string;

   begin  (DeleteFile)
      PrintText ('Delete filename: ',1,22);
      ReadLn (filename);
      if filename[2] <> ':' then if filename[1] <> '\' then
         filename:=DefaultDataDir+'\'+filename;
      if EXIST (filename) then begin
         PrintErrorMsg ('Delete file (y/n)? ',1,19,true,RespCh);
         if (UpCase(RespCh)='Y')
            then begin
               Erase (Name);
            end   (then)
            else begin
               PrintText ('Your Choice? ',1,22); ClrEOL;
            end;   (else)
      end;   (then)
      PrintText ('                    ',1,20);
      PrintText ('Your Choice? ',1,22);
   end;   (DeleteFile)

(*****************************************************************************)
(**                                                                         **)
(**   This procedure is a file translation utility. It will translate a     **)
(**   file from the Device Damage Testing program format to the format      **)
(**   required of this waveform analysis program.                           **)
(**                                                                         **)
(*****************************************************************************)

procedure ImportDDT;

   const
      NumVertPoints = 512;          (* Num vert points on 7912 digitizer *)

   var
      delta     : real;             (* Dist between successive data pts. *)
      i         : integer;          (* A counter variable                *)
      SCALE     : boolean;          (* Scale data to real values?        *)
      DummyChar : char;             (* Reads dummy char information      *)
      VoltsDiv  : real;             (* Volts / div of digitizer          *)
      Transfer  : real;             (* Transfer function of probe        *)
      filename  : string;

   procedure ReadIDStrings;

      type
         CodeRange     = 0..2;
```

APPENDIX A

```
var
   SecondsDiv     : real;
   VMax           : real;
   VMin           : real;
   TMax           : real;
   dummy          : string;
   DiskNum        : integer;
   Code           : integer;
   OKAY           : boolean;

procedure ReadString (VAR StringToRead : string;
                          EndCode      : CodeRange);

   var
      DONE : boolean;         (* Done reading string?              *)

   begin   (ReadString)
      DONE:=false;
      i:=Ø;
      repeat
         i:=succ(i);
         read (Name,StringToRead[i]);
         case EndCode of
            Ø: if StringToRead[i] = ',' then DONE:=true;
            1: if StringToRead[i] = '*' then DONE:=true;
            2: if EOLN (Name)         then DONE:=true;
         end;   (case)
      until DONE;
      StringToRead[Ø]:=char(i-1);
   end;   (ReadString)

begin   (ReadIDStrings)
   ReadString (dummy,Ø);
   Val (dummy,NumPoints,Code);
   NumPoints:=succ(NumPoints);

   read (Name,DummyChar);
   ReadString (dummy,1);
   info[1]:='NAME OF FILE: '+dummy;

   read (Name,DummyChar);
   ReadString (dummy,Ø);
   if dummy[1] = '.' then dummy:='Ø'+dummy;
   Val (dummy,VoltsDiv,Code);
   info[2]:='AMPLITUDE / DIV: '+dummy;

   ReadString (dummy,Ø);
   if dummy[1] = '.' then dummy:='Ø'+dummy;
   Val (dummy,SecondsDiv,Code);
   info[3]:='SEC / DIV: '+dummy;
   delta:=1Ø*SecondsDiv/pred(NumPoints);

   ReadString (dummy,Ø);
   if dummy[1] = '.' then dummy:='Ø'+dummy;
   Val (dummy,VMax,Code);
   info[4]:='MAX AMPLITUDE: '+dummy;

   ReadString (dummy,Ø);
   if dummy[1] = '.'
      then dummy:='Ø'+dummy
   else if ((dummy[1] = '-') AND (dummy[2] = '.'))
      then begin
         dummy[1]:='Ø';
         dummy:='-'+dummy;
      end;   (else if)
```

```
      Val (dummy,TMax,Code);
      info[5]:='TIME OF MAX AMPLITUDE: '+dummy;

      ReadString (dummy,Ø);
      if dummy[1] = '.' then dummy:='Ø'+dummy;
      Val (dummy,VMin,Code);
      info[6]:='MIN AMPLITUDE: '+dummy;

      read (Name,DummyChar);
      ReadString (dummy,1);
      read (Name,DummyChar);
      info[7]:='TIME: '+dummy;

      read (Name,DummyChar);
      ReadString (dummy,1);
      info[8]:='DATE: '+dummy;

      read (Name,DummyChar);
      read (Name,DummyChar);
      ReadString (dummy,1);
      read (Name,DummyChar);
      info[14]:='COMMENTS: '+dummy;

      ReadString (dummy,Ø);
      if dummy[1] = '.' then dummy:='Ø'+dummy;
      Val (dummy,Transfer,Code);
      info[9]:='TRANSFER FUNCTION: '+dummy;

      read (Name,DummyChar);
      ReadString (dummy,1);
      read (Name,DummyChar);
      info[1Ø]:='OTHER: '+dummy;

      ReadLn (Name,DiskNum);
      info[11]:='DISK NUMBER: '+char(DiskNum+ord('Ø'));

      read (Name,DummyChar);
      ReadString (dummy,1);
      read (Name,DummyChar);
      info[12]:='DEVICE UNDER TEST: '+dummy;

      read (Name,DummyChar);
      ReadString (dummy,1);
      info[13]:='PROBE: '+dummy;
   end;   {ReadIDStrings}

var
   s     : string;
   error : integer;

begin   {ImportDDT}
   SCALE:=true;
   PrintText ('Enter DDT filename: ',1,22);
   readln (filename);
   if filename[2] <> ':' then if filename[1] <> '\' then
      filename:=DefaultDataDir+'\'+filename;
   if NOT EXIST (filename)
      then begin
         PrintErrorMsg ('File does not exist!',1,19,false,RespCh);
         PrintText ('Your Choice? ',1,22); ClrEOL;
      end   (then)
      else begin
         TRANS:=false;
         ORIG:=true;
         ACCEPT:=true;
         PrintText ('Scale data to real values? ',1,2Ø);
```

```
            repeat
               DummyChar:=ReadKey;
            until (UpCase(DummyChar) in ['Y','N',ENTER]);
            write (UpCase(DummyChar));
            if (UpCase(DummyChar)='N') then SCALE:=false;
            Assign (Name,filename);
            Reset (Name);
            ReadIDStrings;
            NumFreqs:=NumPoints div 2;
            for i:=Ø to NumPoints-1 do begin
               readln (Name,s);
               if (s[1] = '.')
                  then s:='Ø'+s
               else if ((s[1] = '-') AND (s[2] = '.'))
                  then begin
                     s[1]:='Ø';
                     s:='-'+s;
                  end;   (else if)
               Val (s,ampl^[i],error);
               time^[i]:=i*delta;
               if SCALE then
                  ampl^[i]:=ampl^[i]*(8*VoltsDiv*Transfer/NumVertPoints);
            end;   (for)
         Close (Name);
      end;   (else)
   end;   (ImportDDT)

(*****************************************************************************)
(***                                                                       ***)
(***  This procedure lists the directory of the current (logged) drive.    ***)
(***                                                                       ***)
(*****************************************************************************)

procedure DiskDirectory;

   const
      MaxDirectoryEntries = 2ØØ;
      DefaultDrive        = Ø;
      Search              = $3Ø;      {search for directories and files   }

   type
      MaxEntries  = 1..MaxDirectoryEntries;
      Colors      = 1..8;

   var
      NamR        : array [1..MaxDirectoryEntries] of string [12];
      EntryDir    : array [1..MaxDirectoryEntries] of boolean;
      DefaultDir  : string;
      buffer      : string;
      temp        : SearchRec;
      DirColor    : byte;
      DirBack     : byte;
      EntryNumber : byte;
      NumEntries  : byte;
      NumScreens  : byte;
      x           : byte;
      y           : byte;
      z           : byte;
      ch          : char;
      Drive       : byte;
      CurrMinX    : byte;
      CurrMaxX    : byte;
      CurrMinY    : byte;
      CurrMaxY    : byte;
      temp_int    : longint;
```

```
begin   {DiskDirectory}
   CurrMinX:=succ(Lo(WindMin));
   CurrMinY:=succ(Hi(WindMin));
   CurrMaxX:=succ(Lo(WindMax));
   CurrMaxY:=succ(Hi(WindMax));
   Window (1,1,80,25);
   DirColor:=abs((7-ForeColor) mod 16);
   DirBack :=abs((7-BackColor) mod 16);
   ClrScr;
   GetDir (DefaultDrive,DefaultDir);
   ChDir (DefaultDataDir);
   FillChar (NamR,SizeOf(NamR),0);
   FillChar (buffer,SizeOf(buffer),0);
   buffer[0]:=char(0);
   PrintText ('File mask: ',1,1); readln (buffer);
   if (length(buffer) = 0)
      then begin
         buffer:='*.*';
         if DefaultDataDir[2] = ':'
            then Drive:=ord(UpCase(DefaultDataDir[1]))-ord('A')+1
            else Drive:=DefaultDrive;
         PrintText (DefaultDataDir,12,1);
      end   {then}
      else begin
         if buffer[2] = ':'          {Get drive number}
            then begin
               Drive:=ord(UpCase(buffer[1]))-(ord('A')-1);
               if length(buffer) = 2 then buffer:=buffer+'*.*';
            end   {then}
            else begin
               if DefaultDataDir[2] = ':'
                  then Drive:=ord(UpCase(DefaultDataDir[1]))-ord('A')+1
                  else Drive:=DefaultDrive;
            end;   {else}
      end;   {else}
   EntryNumber:=0;
   FindFirst (buffer,search,temp);
   if (DosError = 0) AND (temp.Attr = Directory) then begin
      {$I-} ChDir (buffer); {$I+}
      if IOResult = 0 then buffer:='*.*';
      FindFirst (buffer,search,temp);
   end;   {if}
   while (DosError = 0) do begin
      EntryNumber:=succ(EntryNumber);
      NamR[EntryNumber]:=temp.Name;
      if temp.attr = Directory
         then EntryDir [EntryNumber]:=true
         else EntryDir [EntryNumber]:=false;
      FindNext (temp);
   end;   {while}
   NumEntries:=EntryNumber;
   NumScreens:=(NumEntries-1) div 72 +1;
   if (NumEntries >= 1)
      then begin
         EntryNumber:=1;
         for z:=1 to NumScreens do begin
            for y:=3 to 20 do
               for x:=1 to 4 do begin
                  GotoXY (20*x-19,y);
                  if EntryDir[EntryNumber]
                     then begin
                        TextColor (DirColor);
                        TextBackground (DirBack);
                     end   {then}
                     else begin
                        TextColor (ForeColor);
```

```
                        TextBackGround (BackColor);
                     end;   (else)
                  write (NamR[EntryNumber]);
                  EntryNumber:=succ(EntryNumber);
               end;   (for)
            if (NumScreens > 1) then if (z < NumScreens) then begin
               PrintText ('Press any key for more entries....',1,24);
               ch:=ReadKey;
               ClrScr;
               PrintText ('File mask: '+buffer,1,1);
            end;   (if)
         end;   (for)
      end   (then)
      else begin
         PrintErrorMsg ('File not found! ',4,5,false,ch);
      end;   (else)
   GotoXY (1,22);
   TextColor (ForeColor);
   TextBackground (BackColor);
   temp_int:=DiskFree (Drive) div 1000;
   if Drive > 0
      then ch:=char(Drive+ord('A')-1)
      else ch:='C';
   write ('Drive ',ch,': has ',temp_int,' kB free.');
   {$I-} ChDir (DefaultDir); {$I+}
   ch:=char(IOResult);                    {Dummy assignment                }
   PrintText ('Press any key to return ...',1,24);
   ch:=ReadKey;
   ClrScr;
   Window (1,1,80,25);
   Window (CurrMinX,CurrMinY,CurrMaxX,CurrMaxY);
end;   (DiskDirectory)

procedure ReadInfoLines;

   var
      ch  : char;
      int : byte;

   begin   (ReadInfoLines)
      Window (1,1,80,25);
      ClrScr;
      GotoXY (1,1);
      for int:=1 to MaxInfo do
         writeln (info[int]);
      PrintText ('Press any key to return to the File I/O menu ...',1,25);
      ch:=ReadKey;
      ClrScr;
      Window (StartColumn-1,2,80,25);
   end;   (ReadInfoLines)

procedure FileIOMenu;

   begin
      (*** Set up window, print FileIO Menu, prompt for choice.          ***)
      ClrScr;
      PrintText ('File I/O Menu',11,1);
      PrintText ('Select Option by typing a number:',1,3);
      PrintText ('1. Save Data to disk          ',9,6);
      PrintText ('2. Retrieve Data from disk    ',9,8);
      PrintText ('3. Delete Data from disk      ',9,10);
      PrintText ('4. Import DDT file from disk  ',9,12);
      PrintText ('5. Disk Directory             ',9,14);
      PrintText ('6. Display Information Lines  ',9,16);
      PrintText ('9. Exit to Main Menu          ',9,18);
      PrintText ('Your choice?                  ',1,22);
```

```
      GotoXY (14,22); ClrEOL;
      IODONE:=false;
   end;   {FileIOMenu}

procedure FileIO;

   begin   {FileIO}
      Window (StartColumn-1,2,80,25);
      IODONE:=false;
      while NOT IODONE do begin
         FileIOMenu;
         repeat
            RespCh:=ReadKey;
         until RespCh in ['1'..'6','9'];
         choice:=ord(RespCh)-ord('0');
         case choice of                                  (* Choice:          *)
            1: SaveFile;                                 (* Save file.       *)
            2: RetrieveFile;                             (* Retrieve file.   *)
            3: DeleteFile;                               (* Delete file.     *)
            4: ImportDDT;                                (* Import file.     *)
            5: DiskDirectory;                            (* Disk Directory.  *)
            6: ReadInfoLines;
            9: IODONE:=true;                             (* Return.          *)
         end;   {case}
      end;   {while}
      Window (1,1,80,25);
      ClrScr;
   end;   {FileIO}

(*****************************************************************************)

begin   {Initialization}
end.   {Initialization}
```

A-3.6 CreateWaveform

Unit CreateWaveform creates one of four kinds of waveforms:

1. damped sine wave,
2. double exponential,
3. reciprocal double exponential, and
4. general exponential sum.

The damped sine wave is defined as

$$g(t) = A\, e^{-\alpha t} \sin(2\pi f t + \theta) \ .$$

The double exponential is defined as

$$g(t) = A(e^{-\alpha t} - e^{-\beta t}) \ .$$

The reciprocal double exponential is defined as

$$g(t) = \frac{A}{e^{\alpha(t - t_0)} - e^{\beta(t - t_0)}} \ .$$

The general exponential sum is defined as

$$g(t) = \sum_{i=1}^{n} A_i e^{\alpha_i t} \ .$$

After the user inputs the various parameters required to create one of these waveforms, he is given the option of adding noise to it. The amount of noise added is defined by the user as a given signal-to-noise ratio, input in decibels (dB).

```
unit CreateWaveform;

interface
uses
   Crt,
   Global;

procedure AnalyticWaveform;


(*****************************************************************************)

implementation

procedure AnalyticWaveform;

   var
      choice         : byte;            (* Choice of waveforms to create     *)
      RespCh         : char;

   procedure AnalyticMenu;

      begin   (AnalyticMenu)
         ClrScr;
         Window (StartColumn-1,2,80,25);
         PrintText ('Create Waveform Menu',9,1);
         PrintText ('Select waveform to create by typing a number.',1,3);
         PrintText ('1. Damped Sine Wave       ',9,6);
         PrintText ('2. EMP Double Exponential ',9,8);
         PrintText ('3. Reciprocal Double Exponential',9,10);
         PrintText ('4. General Exponential Sum',9,12);
         PrintText ('9. Exit to Main Menu      ',9,14);
         PrintText ('Your Choice?              ',1,22);
         GotoXY (14,22);
         repeat
            RespCh:=ReadKey;
         until (RespCh in ['1'..'4','9']);
         choice:=ord(RespCh)-ord('0');
         if choice <> 9 then begin
            NumPoints:=TNArraySize;
            NumFreqs :=TNArraySize div 2;
         end;   (if)
      end;   (AnalyticMenu)

   procedure MakeDampSine;

   var
      i              : integer;
      A              : real;
      alpha          : real;
      t              : real;
      beta           : real;
      cycles         : real;
      phi            : real;
      dummy          : GraphName;

      begin   (MakeDampSine)
         PrintText ('Damped Sine Wave',StartColumn+8,2);
         PrintText ('This wave is of the form   ',1,5);
         write ('A * exp(-alpha*t) * sin(2*PI*f*t + phi).');
         repeat
            PrintText ('Period of wave =           ',1,8);
            GotoXY (18,8); readln (dummy);
            if dummy = '' then dummy:='0';
            String_to_Value (dummy,beta,dummy);
         until (beta <> 0);
```

APPENDIX A

```
            repeat
               PrintText ('Number of cycles =          ',1,10);
               GotoXY (20,10); readln (dummy);
               if dummy = '' then dummy:='0';
               String_to_Value (dummy,cycles,dummy);
            until (cycles <> 0);
            repeat
               PrintText ('A =           ',1,12);
               GotoXY (5,12); readln (dummy);
               if dummy = '' then dummy:='0';
               String_to_Value (dummy,A,dummy);
            until (A <> 0);
            PrintText ('alpha = ',1,14); readln (dummy);
            if dummy = '' then dummy:='0';
            String_to_Value (dummy,alpha,dummy);
            PrintText ('phi (degrees) = ',1,16); readln (dummy);
            if dummy = '' then dummy:='0';
            String_to_Value (dummy,phi,dummy);
            phi:=phi*pi/180;
            for i:=0 to NumPoints-1 do begin
               t:=beta*i*cycles/(NumPoints-1);
               time^[i]:=t;
               ampl^[i]:=A*exp(-alpha*t)*sin(2*PI*t/beta+phi);
            end;   {for}
            TRANS:=false;
            ORIG:=true;
            ACCEPT:=true;
         end;   {MakeDampSine}

      procedure MakeDoubleExp;

         var
            ch          : char;
            i           : integer;
            A           : real;
            alpha       : real;
            t           : real;
            beta        : real;
            timeofmax   : real;
            maxtime     : real;
            dummy       : GraphName;

         begin   {MakeDoubleExp}
            PrintText ('EMP Double Exponential',StartColumn+8,2);
            PrintText ('This wave is of the form   ',1,5);
            write ('A * [exp(-a*t) - exp(-b*t)].');
            repeat
               PrintText ('Maximum amplitude = ',1,8); readln (dummy);
               if dummy = '' then dummy:='0';
               String_to_Value (dummy,A,dummy);
            until (A <> 0);
            repeat
               PrintText ('risetime =           ',1,10);
               GotoXY (12,10); readln (dummy);
               if dummy = '' then dummy:='0';
               String_to_Value (dummy,beta,dummy);
            until (beta > 0);
            repeat
               PrintText ('falltime =            ',1,12);
               GotoXY (12,12); readln (dummy);
               if dummy = '' then dummy:='0';
               String_to_Value (dummy,alpha,dummy);
            until (alpha > 0);
            maxtime:=3*alpha;
            alpha:=ln(10)/alpha;
            beta :=ln(9) /beta;
```

```
      timeofmax:=ln(beta/alpha)/(beta-alpha);
      A:=A/(exp(-alpha*timeofmax)-exp(-beta*timeofmax));
      for i:= Ø to NumPoints-1 do begin
         t:=maxtime*i/(NumPoints-1);
         time^[i]:=t;
         ampl^[i]:=A*(exp(-alpha*t)-exp(-beta*t));
      end;   {for}
      TRANS:=false;
      ORIG:=true;
      ACCEPT:=true;
   end;   {MakeDoubleExp}

procedure MakeRecipDoubleExp;

   var
      i              : integer;
      A              : real;
      alpha          : real;
      b              : real;
      t              : real;
      beta           : real;
      tØ             : real;
      dummy          : GraphName;

   begin   {MakeRecipDoubleExp}
      PrintText ('Reciprocal Double Exponential',StartColumn+8,2);
      PrintText ('This wave is of the form   ',1,5);
      write ('A / (exp[a*(t-tØ)] - exp[b*(t-tØ)]).');
      PrintText ('Initial time = ',1,8); readln (dummy);
      String_to_Value (dummy,tØ,dummy);
      PrintText ('Maximum time = ',1,1Ø); readln (dummy);
      String_to_Value (dummy,beta,dummy);
      PrintText ('A = ',1,12); readln (dummy);
      String_to_Value (dummy,A,dummy);
      PrintText ('a = ',1,14); readln (dummy);
      String_to_Value (dummy,alpha,dummy);
      PrintText ('b = ',1,16); readln (dummy);
      String_to_Value (dummy,b,dummy);
      for i:=Ø to NumPoints-1 do begin
         t:=beta*i/(NumPoints-1);
         time^[i]:=t;
         ampl^[i]:=A/(exp(alpha*(t-tØ))-exp(b*(t-tØ)));
      end;   {for}
      TRANS:=false;
      ORIG:=true;
      ACCEPT:=true;
   end;   {MakeRecipDoubleExp}

procedure MakeGeneralSum;

   var
      i              : integer;
      j              : integer;
      A              : real;
      alpha          : real;
      t              : real;
      beta           : real;
      NumTerms       : integer;
      dummy          : GraphName;

   begin   {MakeGeneralSum}
      PrintText ('General Exponential Sum',StartColumn+8,2);
      PrintText ('This wave is of the form   ',1,5);
      write ('SUM A*exp(a*t).');
      repeat
         PrintText ('Maximum time =                    ',1,8);
```

```
            GotoXY (16,8); readln (dummy);
            if dummy = '' then dummy:='Ø';
            String_to_Value (dummy,beta,dummy);
         until (beta > Ø);
         for i:=Ø to NumPoints-1 do begin
            t:=beta*i/(NumPoints-1);
            time^[i]:=t;
            ampl^[i]:=Ø;
         end;   (for)
         PrintText ('Number of exponential terms:        ',1,1Ø);
         GotoXY (3Ø,1Ø); readln (dummy);
         if dummy = '' then dummy:='Ø';
         String_to_Value (dummy,t,dummy);
         NumTerms:=round(t);
         ORIG:=false;
         for i:=1 to NumTerms do begin
            GotoXY (1,14); ClrEOL;
            GotoXY (1,16); ClrEOL;
            GotoXY (1,12); writeln ('Exponential term #',i);
            PrintText ('A = ',5,14); ClrEOL; readln (dummy);
            if dummy = '' then dummy:='Ø';
            String_to_Value (dummy,A,dummy);
            if NOT ORIG then if A <> Ø then ORIG:=true;
            PrintText ('a = ',5,16); ClrEOL; readln (dummy);
            if dummy = '' then dummy:='Ø';
            String_to_Value (dummy,alpha,dummy);
            for j:=Ø to NumPoints-1 do
               ampl^[j]:=A*exp(alpha*time^[j]) + ampl^[j];
         end;   (for)
         TRANS:=false;
         ACCEPT:=true;
      end;   (MakeGeneralSum)

   procedure AddNoise;

      var
         charchoice    : char;          (* Add noise?                      *)
         dummy         : GraphName;     (* Scale value of noise            *)
         Scale         : real;          (* Scale value of noise            *)
         max_r         : real;          (* Max value of waveform           *)
         max_i         : integer;       (* Index of max value of waveform *)
         i             : integer;       (* Counter variable                *)
         noise         : real;

      begin   (AddNoise)
         PrintText ('Add noise to waveform (y/n)? ',1,2Ø);
         repeat
            charchoice:=ReadKey;
            charchoice:=UpCase(charchoice);
         until (charchoice in ['Y','N',ENTER]);
         if (charchoice = ENTER)
            then write ('N')
            else write (charchoice);
         if (charchoice = 'Y') then begin
            PrintText ('Signal-to-Noise ratio (dB) = ',1,22); read (dummy);
            if dummy = ''
               then Scale:=1ØØØ
               else String_to_Value (dummy,Scale,dummy);
            max_i:=Ø;
            for i:=1 to NumPoints-1 do
               if ampl^[i] > ampl^[max_i] then max_i:=i;
            max_r:=ampl^[max_i];
            Scale:=max_r*exp((-Scale/1Ø)*ln(1Ø));   (converts dB to real #)
            Randomize;
            for i:=Ø to NumPoints-1 do begin
               noise:=Scale*random;
```

```
                ampl^[i]:=ampl^[i]+noise;
            end;   (for)
        end;   (if)
    end;   (AddNoise)

var
    Make          : boolean;          (* Has a waveform been created?      *)

begin   (AnalyticWaveform)
    AnalyticMenu;
    Window (1,1,80,25);
    ClrScr;
    Make:=true;
    case choice of
        1: MakeDampSine;
        2: MakeDoubleExp;
        3: MakeRecipDoubleExp;
        4: MakeGeneralSum;
        9: begin
               Make:=false;
           end;
    end;   (case)
    if Make then AddNoise;
end;   (AnalyticWaveform)

(*****************************************************************************)

begin   (Initialization)
end.   (Initialization)
```

A-3.7 GraphWaveform

Unit GraphWaveform displays a graph of the data. The graph defaults to a linear graph for time-domain data, a log graph for the magnitude of the frequency-domain data, and a log-linear graph for the phase of the frequency-domain data. The following procedures are contained in this file:

1. ClearWindow--clears a portion of the screen.

2. PrintMenu--displays the graph menu.

3. ClearMenu--removes the graph menu.

4. GraphData--graphs the data on the screen.

5. ChangeInterval--changes the endpoints of the interval to be graphed.

6. OutputScreen--prepares the screen to be printed.

7. ChangePlotType--changes to any of the four possible plot types.

8. GraphResults--performs the following:

   a. Initialize system
   b. Repeat

      (1) Option 1: Output Screen
      (2) Option 2: Change Plot Type, Intervals
      (3) Option 3: Reset Parameters
      (4) Option 9: Finished

   until finished.

```
(*****************************************************************************)
(***                                                                       ***)
(***  GraphResults plots either the time domain data or the frequency      ***)
(***  domain data, as determined by the value of parameter PlotType. The   ***)
(***  data is either plotted over the whole interval, or over a user       ***)
(***  defined subinterval. Additionally, a hard copy may be made of the    ***)
(***  current screen, a zoom factor may be given, or the type of plot may  ***)
(***  be changed. Initially the plot is drawn as a log-log plot. Four      ***)
(***  types of plots are defined:                                          ***)
(***       1. Linear x, linear y.                                          ***)
(***       2. Log x,    log y.                                             ***)
(***       3. Log x,    linear y.                                          ***)
(***       4. Linear x, log y.                                             ***)
(***                                                                       ***)
(***                                                                       ***)
(*****************************************************************************)

unit GraphWaveform;

interface
uses
   Crt,
   Graph,
{$U c:\pascal4\graphix\drawgraf.tpu} DrawGraf,
   Global,
   GraphText;

procedure GraphResults (PlotType : char);


(*****************************************************************************)

implementation

procedure GraphResults (PlotType : char);

   const
      NormSize    = 1;

   var
      i           : integer;     (* Loop counter                          *)
      MinFirst    : integer;     (* Smallest point that can be graphed.   *)
      MaxLast     : integer;     (* Largest  point that can be graphed.   *)
      first       : integer;     (* Subinterval setup: graph from         *)
      last        : integer;     (*    first to last points of Temp.      *)
      RightEnd    : real;        (* Subinterval setup: user               *)
      LeftEnd     : real;        (*      inputted endpoints.              *)
      tempx       : real;        (* Temp var for swapping values.         *)
      maximum     : real;        (* Largest value of maximum.             *)
      response    : byte;        (* Choice of options.                    *)
      temp        : real;        (* Variable for swapping values.         *)
      ViewDONE    : boolean;     (* Done viewing graph?                   *)
      Title       : string;      (* Title of graph.                       *)
      XTitle      : integer;     (* X coordinate to print title.          *)
      YTitle      : integer;     (* Y coordinate to print title.          *)
      TitlePos    : integer;     (* Position of 'Title:' in title.        *)
      j           : byte;        (* Temp counter variable.                *)
      Plot        : byte;        (* Plot type.                            *)
      PlotFREQ    : boolean;     (* Frequency data to be plotted?         *)
      PlotMAG     : boolean;     (* Magnitude of frequency data?          *)
      PlotPHASE   : boolean;     (* Phase of frequency data?              *)
      dummy       : char;


   procedure ClearWindow (x_1,y_1,x_2,y_2 : integer);
```

```
      var
         v : ViewPortType;

      begin   {ClearWindow}
         GetViewSettings (v);
         SetViewPort (x_1,y_1,x_2,y_2,ClipOff);
         ClearViewPort;
         with v do
            SetViewPort (x1,y1,x2,y2,Clip);
      end;   {ClearWindow}

   procedure PrintMenu;

      begin   {PrintMenu}
         PrintText ('Options: 1. Print Screen                ',23,22);
         PrintText ('         2. Change Parameters           ',23,23);
         PrintText ('         3. Reset Parameters            ',23,24);
         PrintText ('         9. Exit to Main Menu           ',23,25);
      end;   {PrintMenu}

   procedure ClearMenu;

      var
         x_1,y_1,x_2,y_2 : integer;

      begin   {ClearMenu}
         Text_To_Cart (1, 21,x_1,y_1);
         Text_To_Cart (80,26,x_2,y_2);
         x_2:=pred(x_2);
         y_2:=pred(y_2);
         ClearWindow (x_1,y_1,x_2,y_2);
      end;   {ClearMenu}

(*****************************************************************************)
(***                                                                       ***)
(***  GraphData graphs the data. Initially, time data is graphed on a      ***)
(***  linear (type 1) graph, magnitudes are graphed on log-log (type 2)    ***)
(***  graphs, and phase is plotted on a semi-log (type 3) graph.           ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure GraphData;

      var
         x       : PlotXYPtr;
         y       : PlotXYPtr;
         x1      : real;
         x2      : real;
         y1      : real;
         y2      : real;
         error   : byte;
         ErrStr  : string[3];
         i       : integer;

      begin   {GraphData}
         new (x); new (y);
         for i:=first to last do begin
            x^[i-first+1]:=OutArrayX^[i];
            y^[i-first+1]:=OutArrayY^[i];
         end;   {for}
         SetTextStyle (DefaultFont,VertDir,NormSize);
         leftx:=2*TextHeight ('H');
         SetTextStyle (DefaultFont,HorizDir,NormSize);
         rightx:=7*((GetMaxX+1) div 8);
         topy:=5*(TextHeight ('H') div 2);
         if GraphicsCard <> HercMono
```

```
            then bottomy:=3*((GetMaxY+1) div 4)
            else bottomy:=3*((GetMaxY+1) div 4) - 2*TextHeight ('H');
         SetViewPort (0,0,GetMaxX,GetMaxY,ClipOn);
         ClearWindow (0,0,GetMaxX,GetMaxY);
         DrawGraph (x,y,last-first+1,Plot,
                    leftx,topy,rightx,bottomy,x1,x2,y1,y2,'','',error);
         if error <> 0 then begin
            str (error,ErrStr);
            PrintText ('[Error '+ErrStr+' in drawing routine.]',23,10);
         end;   (if error)
         dispose (x); dispose (y);
      end;   (GraphData)

(*****************************************************************************)
(***                                                                       ***)
(***  ChangeIntervals divides the array to be graphed into subintervals    ***)
(***  from LeftEnd units to RightEnd units, as input by the user.          ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure ChangeIntervals;

      var
         i              : integer;
         LeftEndString  : string;          (* Left  endpoint + multiplier  *)
         RightEndString : string;          (* Right endpoint + multiplier  *)
         dummystring    : string;          (* dummy units string returned  *)
         old_last       : integer;         (* store old right endpoint     *)
         old_first      : integer;         (* store old left  endpoint     *)
         x1,x2,y1,y2    : integer;         (* Window coordinates of menu   *)

      begin   (ChangeIntervals)
         old_last:=last;
         old_first:=first;
         (*** Set up interval to be plotted. ***)
         FillChar (LeftEndString,SizeOf(LeftEndString),0);
         PrintText ('Interval from ',23,22);
         readln (LeftEndString);
         if length(LeftEndString) > 0
            then String_to_Value (LeftEndString,LeftEnd,dummystring)
            else LeftEnd:=0;
         FillChar (RightEndString,SizeOf(RightEndString),0);
         PrintText ('           to ',23,23);
         readln (RightEndString);
         if length (RightEndString) > 0
            then String_to_Value (RightEndString,RightEnd,dummystring)
            else RightEnd:=OutArrayX^[MaxLast];
         if LeftEnd > RightEnd then begin
            tempx:=LeftEnd;
            LeftEnd:=RightEnd;
            RightEnd:=tempx;
         end;   (if)
         if LeftEnd < OutArrayX^[MinFirst] then LeftEnd:=OutArrayX^[MinFirst];
         if RightEnd > OutArrayX^[MaxLast] then RightEnd:=OutArrayX^[MaxLast];
         first:=MinFirst;
         last:=MaxLast;
         i:=MinFirst;
         while ((OutArrayX^[i] <= LeftEnd)  AND (i < MaxLast))  do i:=succ(i);
         first:=i;
         i:=MaxLast;
         while ((OutArrayX^[i] >= RightEnd) AND (i > MinFirst)) do i:=pred(i);
         last:=i;
         if (last-first) < 5 then begin
            PrintText ('Too few points to plot! ',23,25);
            Buzzer;
            last:=old_last;
```

```
            first:=old_first;
         end;   (then)
         ClearMenu;
      end;   (ChangeIntervals)

(*****************************************************************************)
(***                                                                       ***)
(***  OutputScreen sets up the screen for a hard copy, then calls the      ***)
(***  PrintScreen routine to produce the hard copy.                        ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure OutputScreen;

      var
         PlotX  : integer;            (* coordinates to draw the units at  *)
         PlotY  : integer;            (* coordinates to draw the units at  *)
         labels : string;
         T      : TextSettingsType;

      begin   (Outputscreen)
         ClearMenu;
         PrintText ('Horizontal units: ',1,23);
         readln (info[MaxInfo-3]);
         PlotX:=(rightx+leftx) div 2 + 10*TextWidth ('W');
         PlotY:=bottomy+2;
         OutTextXY (PlotX,PlotY,info[MaxInfo-3]);
         PrintText (' Vertical  units: ',1,24);
         GetTextSettings (T);
         readln (info[MaxInfo-2]);
         SetUserCharSize (1,3,1,2);
         SetTextStyle (DefaultFont,VertDir,NormSize);
         SetTextJustify (CenterText,CenterText);
         PlotX:=leftx-TextHeight ('H');
         PlotY:=(bottomy+topy) div 2;
         OutTextXY (PlotX,PlotY,info[MaxInfo-2]);
         SetTextStyle (DefaultFont,HorizDir,NormSize);
         SetTextJustify (T.Horiz,T.Vert);
         ClearMenu;
         PrintText ('Title of graph: ',1,23);
         readln (info[MaxInfo-1]);
         PrintText ('Sub-Title:      ',1,24);
         readln (info[MaxInfo]);
         ClearMenu;
         XTitle:=40-(length(info[MaxInfo-1]) div 2);
         PrintText (info[MaxInfo-1],XTitle,1);
         XTitle:=40-(length(info[MaxInfo]) div 2);
         PrintText (info[MaxInfo],XTitle,2);
         Print_Screen;
         info[MaxInfo-1]:='Title:        '+info[MaxInfo-1];
         info[MaxInfo]  :='Sub-Title:    '+info[MaxInfo];
         PrintMenu;
      end;   (OutputScreen)

(*****************************************************************************)
(***                                                                       ***)
(***  ChangePlotType changes the plot number or the interval endpoints.    ***)
(***  It is assumed that the window boundaries are already drawn.          ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure ChangePlotType;

      var
         ch         : char;
         NewPlotNum : byte;
```

```
   begin   {ChangePlotType}
      SetViewPort (Ø,Ø,GetMaxX,GetMaxY,ClipOn);
      ClearMenu;
      PrintText ('Option:  1. Linear Plot          ',23,21);
      PrintText ('         2. Log Plot             ',23,22);
      PrintText ('         3. Log-Linear Plot      ',23,23);
      PrintText ('         4. Linear-Log Plot      ',23,24);
      PrintText ('         5. Change Interval      ',23,25);
      repeat
         ch:=ReadKey;
      until (ch in ['1'..'5']);
      NewPlotNum:=ord(ch)-ord('Ø');
      ClearMenu;
      if (NewPlotNum in [1..4])
         then Plot:=NewPlotNum
      else if (NewPlotNum = 5)
         then ChangeIntervals;
      if PlotPHASE then
         if Plot = Log
            then Plot:= LogLin
         else if Plot = LinLog
            then Plot:= Linear;
      GraphData;
      PrintMenu;
   end;   {ChangePlotType}


begin   {GraphResults}
   PlotMAG  :=(UpCase(PlotType) = 'M');
   PlotPHASE:=(UpCase(PlotType) = 'P');
   PlotFREQ :=PlotMAG OR PlotPHASE;
   if PlotFREQ
      then begin
         MinFirst:=Ø;
         MaxLast :=NumFreqs-1;
         if PlotMAG
            then Plot:=Log
            else Plot:=LogLin;
         for i:=MinFirst to MaxLast do begin
            OutArrayX^[i]:=freq^[i];
            if PlotMAG
               then OutArrayY^[i]:=mag^[i]
               else OutArrayY^[i]:=phase^[i]*18Ø/PI;
         end;   (for)
      end   {then}
      else begin
         MinFirst:=Ø;
         MaxLast :=NumPoints-1;
         for i:=MinFirst to MaxLast do begin
            OutArrayX^[i]:=time^[i];
            OutArrayY^[i]:=ampl^[i];
         end;   (for)
         Plot:=Linear;
      end;   {else}
   first:=MinFirst;
   last :=MaxLast;
   InitGraph (GraphicsCard,GraphicsMode,'');
   SetViewPort (Ø,Ø,GetMaxX,GetMaxY,ClipOn);
   if GraphicsCard <> HercMono
      then SetColor (White);
   GraphData;
   ViewDone:=false;
   PrintMenu;
   repeat
      repeat
```

```
            dummy:=ReadKey;
          until (dummy in ['1'..'3','9']);
          response:=ord(dummy)-ord('0');
          case response of
             1: OutputScreen;
             2: ChangePlotType;
             3: begin
                   first:=MinFirst;
                   last :=MaxLast;
                   if PlotMAG
                      then Plot:=Log
                   else if PlotPHASE
                      then Plot:=LogLin
                      else Plot:=Linear;
                   GraphData;
                   PrintMenu;
                end;
             9: ViewDONE:=true;
          end;   {case}
       until ViewDONE;
       CloseGraph;
       TextColor (ForeColor);
       TextBackGround (BackColor);
       ClrScr;
    end;   {GraphResults}

(*****************************************************************************)

begin   {Initialization}
end.   {Initialization}
```

A-3.8 DigitizeWaveform

Unit DigitizeWaveform contains routines to digitize a waveform, using the asynchronous routines of file Async (see sect. A-3.11) to talk to the digitizer through the COM port. The procedures in this file are as follows:

1. Rotate_Waveform--rotates a waveform from the user's coordinate system to the digitizer's coordinate system.

2. Draw_Waveform--draws the digitized wave.

3. Initialize_Digitizer--initializes the digitizer.

4. Read_Digitizer--reads a point from the digitizer.

5. Read_Keyboard--scans keyboard input for either an ESCAPE or ENTER being pressed.

6. Set_Up_Axes--uses first four input points to create *x*- and *y*-axes.

7. Add_Comments--adds comments at the end of the data file. These comments are

   a. test point,
   b. test type,
   c. attenuation factor,
   d. probe number,
   e. *x*-axis units,
   f. *y*-axis units, and
   g. additional comments.

These particular comment fields were chosen to suit the author's purposes; they may be changed without affecting the operation of the program.

APPENDIX A

```
(*****************************************************************************)
(***                                                                       ***)
(***  Digitizer is to read data directly from the digitizing pad, then     ***)
(***  perform analysis on that data until it represents the "actual"       ***)
(***  waveform in the photograph. The digitizer used is the Micro          ***)
(***  Digi-Pad 6"x6" Electromagnetic Type 7 digitizer from GTCO Corp.      ***)
(***  The input parameter DataToDigitize determines whether to digitize    ***)
(***  time or frequency domain data.                                       ***)
(***                                                                       ***)
(*****************************************************************************)

unit DigitizeWaveform;

interface
uses
   Crt,
   Async,
   Graph,
   DrawGraf,
   Global,
   GraphText;

procedure Digitizer ( DataToDigitize : char);


(*****************************************************************************)

implementation

procedure Digitizer ( DataToDigitize : char);

   const
      Reset_Command     = 'RS';        (* Reset digitizer to power-up       *)
      Point_Command     = 'P1';        (* Set digitizer to point input      *)
      parity            = 'N';          (* Set digitizer to no parity       *)
      baud_rate         = 9600;         (* Set digitizer to 9600 Baud       *)
      data_bits         = 8;            (* Set digitizer to 8 bits per word *)
      stop_bits         = 2;            (* Set digitizer to two stop bits   *)
      PauseLength       = 250;          (* Pause after sending commands     *)
      ShortPauseLength  = 10;           (* Delay between digitizer bytes    *)
      MinX              = 100;          (* Left edge of digitizer work area *)
      MaxX              = 2000;         (* Right edge of digitizer work area*)
      MinY              = 200;          (* Bottom edge of digitizer work area*)
      MaxY              = 1800;         (* Top edge of digitizer work area  *)
      NormSize : byte   = 1;            (* Normal size of default font      *)

   var
      ch1,ch2,ch3,ch4,ch5 : char;       (* input from digitizer             *)
          in2,in3,in4,in5 : integer;    (* integer (ch)                     *)
      STOP                : boolean;    (* done yet?                        *)
      i                   : integer;    (* counter variable                 *)
      INCOMING            : boolean;    (* valid data coming from digitizer? *)
      EMPTY               : boolean;    (* is input buffer empty?           *)
      VALID_PORT          : boolean;    (* is a device connected to COM?    *)
      quit                : char;       (* done viewing picture?            *)
      min_time            : real;       (* minimum time                     *)
      max_time            : real;       (* maximum time                     *)
      min_amplitude       : real;       (* minimum amplitude                *)
      max_amplitude       : real;       (* maximum amplitude                *)
      horiz_units         : string;     (* horizontal axis units            *)
      vert_units          : string;     (* vertical axis units              *)

(*****************************************************************************)
(***                                                                       ***)
(***  Rotate_Waveform transforms each point (x',y') in oblique             ***)
(***  coordinates to point (x,y) in rectangular coordinates. This is       ***)
```

```
(***  done to ensure true horizontal and vertical axes. The specific      ***)
(***  steps performed are                                                  ***)
(***      1. Calculate true origin; set equal to (x1,y1).                  ***)
(***      2. Determine scaling factors and units.                          ***)
(***      3. Transform oblique coordinates to rectangular coordinates.     ***)
(***      4. Eliminate coordinate axes information.                        ***)
(***      5. Eliminate non-sequential data points.                         ***)
(***                                                                       ***)
(*************************************************************************)

   procedure Rotate_Waveform;

      const
         epsilon = 1;

      var
         xØ,yØ                 : real;     (* coordinates of true origin   *)
         mr                    : real;     (* slope of radius axis         *)
         mx                    : real;     (* slope of digitized x axis    *)
         my                    : real;     (* slope of digitized y axis    *)
         r                     : real;     (* magnitude of radius vector   *)
         rx                    : real;     (* length of digitized x axis   *)
         ry                    : real;     (* length of digitized y axis   *)
         theta                 : real;     (* phase of radius vector       *)
         phi                   : real;
         omega                 : real;
         i                     : integer;  (* counter variable             *)
         j                     : integer;  (* counter variable             *)
         k                     : integer;  (* counter variable             *)
         dummy                 : string;   (* time plus multiplier, units  *)
         total_amplitude       : real;     (* amp/div * number of div      *)
         xfactor               : real;     (* x scaling factor             *)
         yfactor               : real;     (* y scaling factor             *)
         delta_x               : real;     (* diff between x coor, origin  *)
         delta_y               : real;     (* diff between y coor, origin  *)

      begin   (Rotate_Waveform)
         (***    Calculate origin   ***)
         mx:=(TempYPtr^[2]-TempYPtr^[1])/(TempXPtr^[2]-TempXPtr^[1]);
         if (abs (TempXPtr^[4]-TempXPtr^[3]) < epsilon)
            then xØ:=(TempXPtr^[3]+TempXPtr^[4])/2
            else begin
               my:=(TempYPtr^[4]-TempYPtr^[3])/(TempXPtr^[4]-TempXPtr^[3]);
               xØ:=(TempYPtr^[2]-TempYPtr^[3]+my*TempXPtr^[3]
                                             -mx*TempXPtr^[2])/(my-mx);
            end;   (else)
         yØ:=TempYPtr^[2]+mx*(xØ-TempXPtr^[2]);
         TempXPtr^[Ø]:=xØ;
         TempYPtr^[Ø]:=yØ;

         (***   Determine scaling factors   ***)
         delta_x:=TempXPtr^[2]-xØ;
         delta_y:=TempYPtr^[2]-yØ;
         rx:=sqrt(sqr(delta_x)+sqr(delta_y));
         delta_x:=TempXPtr^[4]-TempXPtr^[3];
         delta_y:=TempYPtr^[4]-TempYPtr^[3];
         ry:=sqrt(sqr(delta_x)+sqr(delta_y));

         if (UpCase(DataToDigitize) = 'F')
            then begin
               PrintText ('Minimum frequency: ',22,21);
               read (dummy);
               String_to_Value (dummy,min_time,horiz_units);
               PrintText ('Maximum frequency: ',22,21); ClrEOL;
               GotoXY (41,21); read (dummy);
               String_to_Value (dummy,max_time,horiz_units);
```

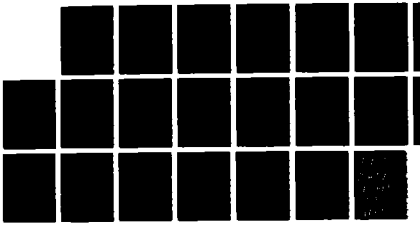
```
      xfactor:=(max_time-min_time)/rx;
      PrintText ('Minimum amplitude:          ',22,21);
      GotoXY (41,21); read (dummy);
      String_to_Value (dummy,min_amplitude,vert_units);
      PrintText ('Maximum amplitude:          ',22,21);
      GotoXY (41,21); read (dummy);
      String_to_Value (dummy,max_amplitude,vert_units);
      yfactor:=(max_amplitude-min_amplitude)/ry;
   end   {then}
   else begin
      min_time:=0;
      PrintText ('Maximum time: ',22,21); ClrEOL;
      read (dummy);
      String_to_Value (dummy,max_time,horiz_units);
      xfactor:=max_time/rx;
      min_amplitude:=0;
      PrintText ('Total amplitude:            ',22,21);
      GotoXY (39,21); read (dummy);
      String_to_Value (dummy,max_amplitude,vert_units);
      yfactor:=max_amplitude/ry;
   end;   {else}
PrintText ('                                          ',22,21);

(*** Switch from input coordinates to rectangular coordinates    ***)
delta_x:=TempXPtr^[2]-x0;
delta_y:=TempYPtr^[2]-y0;
theta:=arctan (delta_y/delta_x);
delta_y:=TempYPtr^[3]-y0;
if delta_y < epsilon
   then begin
      delta_y:=TempYPtr^[4]-y0;
      delta_x:=TempXPtr^[4]-x0;
      if delta_x < epsilon
         then phi:=-PI/2
         else phi:=arctan (delta_y/delta_x);
   end   {then}
   else begin
      delta_x:=TempXPtr^[3]-x0;
      if delta_x < epsilon
         then phi:=PI/2
         else phi:=arctan (delta_y/delta_x);
   end;   {else}
omega:=phi-theta;
for i:=5 to NumPoints + 4 do begin
   delta_x:=(TempXPtr^[i]-x0)*cos(theta)+
            (TempYPtr^[i]-y0)*cos(omega+theta);
   delta_y:=(TempXPtr^[i]-x0)*sin(theta)+
            (TempYPtr^[i]-y0)*sin(omega+theta);
   TempXPtr^[i]:=(delta_x*xfactor + min_time);
   TempYPtr^[i]:=(delta_y*yfactor + min_amplitude);
end;   {for}

(***   Eliminate coordinate axes, non-sequential information.   ***)
if (UpCase(DataToDigitize) = 'F')
   then begin
      TempXPtr^[0]:=TempXPtr^[5];
      TempYPtr^[0]:=TempYPtr^[5];
      i:=1;
      j:=6;
   end   {then}
   else begin
      TempXPtr^[0]:=0;
      TempYPtr^[0]:=0;
      TempXPtr^[1]:=TempXPtr^[5];
      TempYPtr^[1]:=TempYPtr^[5];
      i:=2;
```

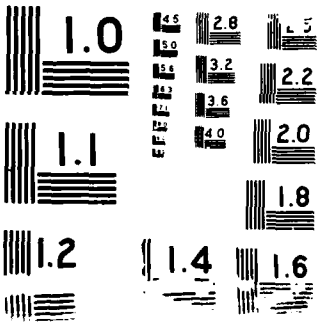
```
            j:=6;
            inc (NumPoints,1);
         end;   {else}
      k:=0;
      repeat
         while TempXPtr^[j] <= TempXPtr^[i-1] do begin
            inc (j,1);
            inc (k,1);
         end;   {while}
         TempXPtr^[i]:=TempXPtr^[j];
         TempYPtr^[i]:=TempYPtr^[j];
         inc (i,1);
         inc (j,1);
      until (i+k=NumPoints);
      dec (NumPoints,k);
   end;   {Rotate_Waveforms}


(*****************************************************************************)
(***                                                                       ***)
(***  Draw_waveform sets up a window for drawing the digitized waveform,   ***)
(***  then draws it.                                                       ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure Draw_Waveform;

      var
         x     : PlotXYPtr;
         y     : PlotXYPtr;
         x1    : real;
         x2    : real;
         y1    : real;
         y2    : real;
         error : byte;
         i     : integer;

      begin   {Draw_Waveform}
         new (x); new (y);
         ClearViewPort;
         leftx:=GetMaxX div 8;
         rightx:=7*leftx;
         bottomy:=3*(GetmaxY div 8);
         topy:=7*(bottomy div 3);
         for i:=1 to NumPoints do begin
            x^[i]:=TempXPtr^[i];
            y^[i]:=TempYPtr^[i];
         end;   {for}
         DrawGraph (x,y,NumPoints,1,leftx,topy,rightx,bottomy,
                                    x1,x2,y1,y2,'','',error);
         dispose (x); dispose (y);
      end;   {Draw_Waveform}

(*****************************************************************************)
(***                                                                       ***)
(***  Initialize_Digitizer opens the COM port (port SerialPort, as         ***)
(***  selected from the Advanced Options Menu; the default is COM1:)       ***)
(***  then sends commands to the digitizer to reset it to power up         ***)
(***  conditions, then set it to the point mode. The input buffer is       ***)
(***  then emptied.                                                        ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure Initialize_Digitizer;

      begin   {Initialize_Digitizer}
```

```
        VALID_PORT:=true;
        Async_Init;
        if NOT Async_Open (SerialPort,baud_rate,parity,data_bits,stop_bits)
           then begin
              write ('Invalid port!');
              VALID_PORT:=false;
           end   (then)
           else begin
              Async_Send_String (Reset_Command);
              Async_Send_String (Point_Command);
              Delay (PauseLength);
              repeat until NOT Async_Buffer_Check (ch1);
           end;   (else)
     end;   (Initialize_Digitizer)

(*****************************************************************************)
(***                                                                       ***)
(***  Read_Digitizer tests if there is a character in the input buffer.    ***)
(***  If yes, then the next four characters are read. (The Digi-Pad        ***)
(***  digitizer uses five bytes per input point.) The input buffer is      ***)
(***  then emptied. Note that a delay MUST occur between each buffer       ***)
(***  check, or the computer will not recognize the information.           ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure Read_Digitizer;

      begin   (Read_Digitizer)
         if Async_Buffer_Check (ch1) then begin
            Delay (ShortPauseLength);
            if Async_Buffer_Check (ch2) then in2:=integer(ch2) AND $3F;
            Delay (ShortPauseLength);
            if Async_Buffer_Check (ch3) then in3:=integer(ch3) AND $3F;
            Delay (ShortPauseLength);
            if Async_Buffer_Check (ch4) then in4:=integer(ch4) AND $3F;
            Delay (ShortPauseLength);
            if Async_Buffer_Check (ch5) then in5:=integer(ch5) AND $3F;
            inc (i,1);
            TempXPtr^[i]:=(in3 shl 6) OR in2;
            TempYPtr^[i]:=(in5 shl 6) OR in4;
            PutPixel (round(TempXPtr^[i]),round(TempYPtr^[i]),white);
            repeat until NOT Async_Buffer_Check (ch1);
         end;   (if)
      end;   (Read_Digitizer)

(*****************************************************************************)
(***                                                                       ***)
(***  Read_Keyboard detects either an ENTER or ESCAPE key being pressed,   ***)
(***  otherwise, keyboard input is ignored.                                ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure Read_Keyboard;

      begin   (Read_Keyboard)
         if keypressed then begin
            ch1:=ReadKey;
            if ch1 = #13
               then STOP:=true
               else if ch1 = #27 then begin
                  PutPixel (round(TempXPtr^[i]),round(TempYPtr^[i]),black);
                  dec (i,1);
                  SetColor (DrawColor);
               end;   (else)
         end;   (if)
      end;   (Read_Keyboard)
```

AD-A194 249 WAVEFORM ANALYSIS ON THE IBM PC(U) HARRY DIAMOND LABS 2/2
ADELPHI MD J J FALTER APR 88 HDL-TM-88-7

UNCLASSIFIED F/G 12/5 NL




1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.8
3.2
3.6
4.0
4.5
5.0

```
(*****************************************************************************)
(***                                                                       ***)
(***  Set_Up_Axes uses the first two points input as the left and right    ***)
(***  enpoints of the x axis, and the next two points as the top and       ***)
(***  bottom of the y axis.                                                ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure Set_Up_Axes;

      var
         correct : char;
         OKAY    : boolean;

      begin   {Set_Up_Axes}
         repeat
            PrintText ('Enter x axis coordinates: ',22,21);
            i:=0;
            repeat
               Read_Digitizer;
               Read_Keyboard;
            until i=2;
            Line (round(TempXPtr^[1]),round(TempYPtr^[1]),
                  round(TempXPtr^[2]),round(TempYPtr^[2]));
            PrintText ('Enter y axis coordinates: ',22,21);
            repeat
               Read_Digitizer;
               Read_Keyboard;
            until i=4;
            Line (round(TempXPtr^[3]),round(TempYPtr^[3]),
                  round(TempXPtr^[4]),round(TempYPtr^[4]));
            PrintText ('Are axes correct?         ',22,21);
            Buzzer;
            correct:=ReadKey;
            if (UpCase(correct) = 'Y')
               then OKAY:=true
               else begin
                  OKAY:=false;
                  ClearViewport;
                  SetViewPort (MinX,MinY,MaxX,MaxY,ClipOn);
                  STOP:=false;
               end;   {else}
         until OKAY;
      end;   {Set_Up_Axes}

(*****************************************************************************)
(***                                                                       ***)
(***  Add_Comments is used to store descriptive information with the data  ***)
(***  as comments. These descriptors include                               ***)
(***     1. Test Point.                                                    ***)
(***     2. Test Type.                                                     ***)
(***     3. Attenuation factor.                                            ***)
(***     4. Probe Number.                                                  ***)
(***     5. x-axis units.                                                  ***)
(***     6. y-axis units.                                                  ***)
(***     7, Any additional comments.                                       ***)
(***                                                                       ***)
(*****************************************************************************)

   procedure Add_Comments;

      var i : integer;

      begin
         PrintText ('Test Point:    ',22,21); read (info [1]);
```

```
        PrintText ('Test Type:     ',22,22); read (info [2]);
        PrintText ('Attenuation:   ',22,23); read (info [3]);
        PrintText ('Probe Number:  ',22,24); read (info [4]);
        PrintText ('Comments:      ',22,25); read (info [7]);
        info[1]:='Test Point:    '+info[1];
        info[2]:='Test Type:     '+info[2];
        info[3]:='Attenuation:   '+info[3];
        info[4]:='Probe Number:  '+info[4];
        info[5]:='X-Axis units:  '+horiz_units;
        info[6]:='Y-Axis units:  '+vert_units;
        info[7]:='Comments:      '+info[7];
        for i:=8 to MaxInfo do info[i]:=blank;
     end;   (Add_Comments)

(*****************************************************************************)

  begin   (Digitizer)
     InitGraph (GraphicsCard,GraphicsMode,'');
     SetTextStyle (TriplexFont,HorizDir NormSize);
     SetViewPort (MinX,MinY,MaxX,MaxY,ClipOn);
     SetColor (DrawColor);
     STOP:=false;
     Initialize_Digitizer;
     if VALID_PORT then begin
        repeat until NOT Async_Buffer_Check (ch1);
        while keypressed do ch1:=ReadKey;
        Set_Up_Axes;
        PrintText ('Enter x,y coordinate pairs.            ',22,21);
        PrintText ('Press ESC to erase, or ENTER to finish.',22,23);
        repeat
           Read_Digitizer;
           Read_Keyboard;
        until STOP;
        PrintText ('                                       ',22,21);
        PrintText ('                                       ',22,23);
        NumPoints:=i-4;
        if (NumPoints >= 5)
           then begin
              Rotate_Waveform;
              Draw_Waveform;
              Async_Send_String (Reset_Command);
              Async_Close;
              Add_Comments;
              PrintText ('Strike any key to return to Main Menu.',22,20);
              ORIG:=true;
              TRANS:=false;
              ACCEPT:=false;
              time^:=TempXPtr^;
              ampl^:=TempYPtr^;
           end   (then)
           else begin
              PrintText ('Too few points input. ',22,21);
              PrintText ('Strike any key to continue ...',22,23);
              ORIG:=false;
              TRANS:=false;
              ACCEPT:=false;
           end;   (else)
        Quit:=ReadKey;
     end;   (if VALID_PORT)
     RestoreCRTMode;
     TextColor (ForeColor);
     TextBackground (BackColor);
     ClrScr;
  end;   (Digitizer)

(*****************************************************************************)
```

```
begin   (Initialization)
end.    (Initialization)
```

A-3.9 DrawGraf

Unit DrawGraf contains routines which draw a waveform in a specified area of the screen. Four types of graphs may be produced:

1. Linear *x*-axis, linear *y*-axis.

2. Log *x*-axis, log *y*-axis.

3. Log *x*-axis, linear *y*-axis.

4. Linear *x*-axis, log *y*-axis.

The data to be plotted are contained in the arrays pointed to by *x* and *y*. The first array contains the *x*-axis information, and the second contains the corresponding *y*-axis information.

The following are the main procedures and functions contained in this file:

1. Drop--drops nonpositive points on a logarithmic axis.

2. MinMaxLin--finds minimum and maximum values for a linear axis.

3. MinMaxLog--finds minimum and maximum values for a logarithmic axis.

4. Skip--determines how many tic marks to skip when writing numbers on axes.

5. LinearAxis--draws a linear axis.

6. LogAxis--draws a logarithmic axis.

7. LinearScale--scales data for a linear graph.

8. LogScale--scales data for a logarithmic graph.

9. Graph--graphs data.

In addition, when program FFT is started, this unit checks the type of graphics hardware installed. If the hardware is incompatible with that for which the program is configured, then an appropriate error message is displayed and the program is aborted.

A limited amount of error checking is also performed. Currently, only two error messages are defined. These are

0: no error;

1: zero or negative value on log scale.

APPENDIX A

```
unit DrawGraf;

interface
uses
   graph,
   Crt,
   GraphText;

const
   MaxPlotPoints = 2048;
   Linear        = 1;
   Log           = 2;
   LogLin        = 3;
   LinLog        = 4;

type
   real          = extended;
   PlotXYArray   = array [1..MaxPlotPoints] of real;
   PlotXYPtr     = ^PlotXYArray;

procedure DrawGraph (     x             : PlotXYPtr; (* x data to graph     *)
                          y             : PlotXYPtr; (* y data to graph     *)
                          NumPlotPoints : integer;   (* num points in x,y   *)
                          Plot          : byte;      (* type of plot        *)
                          left_x        : integer;   (* left window edge    *)
                          top_y         : integer;   (* top window edge     *)
                          right_x       : integer;   (* right window edge   *)
                          bottom_y      : integer;   (* bottom window edge  *)
                      VAR minx          : real;
                      VAR maxx          : real;
                      VAR miny          : real;
                      VAR maxy          : real;
                          horiz_units   : string;    (* Horiz axis units    *)
                          vert_units    : string;    (* Vert  axis units    *)
                      VAR error         : byte       (* error routine       *)
                     );

{ error = 0   : no error                                 }
{         1   : zero or negative value on log scale      }



(*************************************************************************)

implementation

const
   ln10            = 2.302585092994046;
   MajorTic        = 8;
   MinorTic        = 4;
   MinIntervals    = 2;
   MaxIntervals    = 10;
   MaxDecades      = 5;
   char_size       = 4;

   Power           : string [10]
                   = '1000000000';

   LogTable        : array [2..9] of real
                   = (0.3010,
                      0.4771,
                      0.6021,
                      0.6990,
                      0.7782,
                      0.8451,
```

```
                    0.9031,
                    0.9542);

type
   SkipType        = 1..5;
   WrkString       = string [20];
   WorldType       = record
                       x_1  : real;    (* minimum value of x in window   *)
                       sx_1 : integer;   (* screen coordinate for x_1     *)
                       y_1  : real;    (* minimum value of y in window   *)
                       sy_1 : integer;   (* screen coordinate for y_1     *)
                       x_2  : real;    (* maximum value of x in window   *)
                       sx_2 : integer;   (* screen coordinate for x_2     *)
                       y_2  : real;    (* maximum value of y in window   *)
                       sy_2 : integer;   (* screen coordinate for y_2     *)
                     end;   (record)

var
   XLinear         : boolean;       (* is x axis linear?                 *)
   YLinear         : boolean;       (* is y axis linear?                 *)
   i               : integer;       (* counter variable                  *)
   start           : integer;       (* first nonnegative point in array  *)
   ScaleX          : real;          (* scale x data to proper range      *)
   ScaleY          : real;          (* scale y data to proper range      *)
   FactorX         : integer;       (* increment on linear x axis        *)
   FactorY         : integer;       (* increment on linear y axis        *)
   NumIntervalsX   : byte;          (* num of x intervals to plot        *)
   PosIntervalsX   : byte;          (* num of positive x intervals       *)
   NegIntervalsX   : byte;          (* num of negative x intervals       *)
   NumIntervalsY   : byte;          (* num of y intervals to plot        *)
   PosIntervalsY   : byte;          (* num of positive y intervals       *)
   NegIntervalsY   : byte;          (* num of negative y intervals       *)
   NumDecadesX     : byte;          (* num of x decades to plot          *)
   NumDecadesY     : byte;          (* num of y decades to plot          *)
   PlotX           : integer;       (* x coordinate to draw char at      *)
   PlotY           : integer;       (* y coordinate to draw char at      *)
   temp            : real;          (* temp var for swapping coordinates *)
   ZeroX           : integer;       (* x coordinate of graph's origin    *)
   ZeroY           : integer;       (* y coordinate of graph's origin    *)
   SkipX           : SkipType;      (* how many x tic marks to skip      *)
   SkipY           : SkipType;      (* how many y tic marks to skip      *)
   world           : Worldtype;     (* limits of values to be graphed    *)
   char_height     : shortint;
   char_width      : shortint;

procedure SetUpWorld (x1, y1, x2, y2  : real;
                      sx1,sy1,sx2,sy2 : integer);

   begin   (SetUpWorld)
      with world do begin
         x_1:=x1;   sx_1:=sx1;
         y_1:=y1;   sy_1:=sy1;
         x_2:=x2;   sx_2:=sx2;
         y_2:=y2;   sy_2:=sy2;
      end;   (with)
   end;   (SetUpWorld)

function WhereX (x:real) : integer;     (* finds screen coordinate of x *)

   begin   (WhereX)
      with world do
         WhereX:=sx_1+round(((x-x_1)/(x_2-x_1))*(sx_2-sx_1));
   end;   (WhereX)

function WhereY (y:real) : integer;     (* finds screen coordinate of y *)
```

```
   begin   (WhereY)
      with world do
         WhereY:=sy_1+round(((y-y_1)/(y_2-y_1))*(sy_2-sy_1));
   end;   (WhereY)

procedure DrawLine (x1, y1, x2, y2 : real);

   var
      sx1 : integer;                  (* x1 in screen coordinates          *)
      sx2 : integer;                  (* x2 in screen coordinates          *)
      sy1 : integer;                  (* y1 in screen coordinates          *)
      sy2 : integer;                  (* y2 in screen coordinates          *)

   begin   (DrawLine)
      sx1:=WhereX(x1);
      sy1:=WhereY(y1);
      sx2:=WhereX(x2);
      sy2:=WhereY(y2);
      Line (sx1,sy1,sx2,sy2);
   end;   (DrawLine)

function log10 (x:real) : real;

   begin
      log10:=ln(x)/ln10;
   end;

function exp10 (x:real) : real;

   begin
      exp10:=exp(x*ln10);
   end;

procedure Drop (     x      : PlotXYPtr;
                     y      : PlotXYPtr;
                     Plot   : byte;
                VAR start : integer);

   var
      i     : integer;               (* number of nonpositive x elements  *)
      j     : integer;               (* number of nonpositive y elements  *)

   begin   (Drop)
      start:=1;
      i:=1;
      j:=1;
      if NOT XLinear then
         while x^[i] <= 0 do inc(i,1);
      if NOT YLinear then
         while y^[j] <= 0 do inc(j,1);
      case Plot of
         1: start:=1;
         2: if i >= j
               then start:=i
               else start:=j;
         3: start:=i;
         4: start:=j;
      end;   (case)
   end;   (Drop)

procedure MinMaxLin ( VAR min          : real;
                      VAR max          : real;
                      VAR Scale        : real;
                      VAR Factor       : integer;
                      VAR NumIntervals : byte;
                      VAR PosIntervals : byte;
```

```
                VAR NegIntervals : byte
              );

const
   MaxPower  = 21;
   StepSize  : array [1..9] of integer
             = (1,2,5,10,20,50,100,200,500);

var
   OKAY      : boolean;          (* Done with loop yet?              *)
   i         : integer;          (* Counter variable                 *)
   delta     : real;             (* Max - min                        *)
   MinFudge  : boolean;          (* Fudge factor for min computations *)
   MaxFudge  : boolean;          (* Fudge factor for max computations *)

begin   (MinMaxLin)
   delta:=abs(max-min);
   i:=MaxPower+3;
   repeat
      dec (i,3);
      Scale:=exp10(i);
   until ((5 <= (delta/Scale)) AND ((delta/Scale) < 5000));
   MinFudge:=false;
   MaxFudge:=false;
   if (abs(min/Scale) > MaxInt) then begin
      min:=min/1000;
      MinFudge:=true;
   end;   (if)
   if (abs(max/Scale) > MaxInt) then begin
      max:=max/1000;
      MaxFudge:=true;
   end;   (if)
   if min < 0
      then min:=pred(trunc(min/Scale))
      else min:=     trunc(min/Scale);
   if max < 0
      then max:=     trunc(max/Scale)
      else max:=succ(trunc(max/Scale));
   if max < 0
      then max:=0
   else if min > 0
      then min:=0;
   if MinFudge then min:=min*1000;
   if MaxFudge then max:=max*1000;

   i:=1;
   NumIntervals:=10;
   OKAY:=false;
   repeat
      Factor:=StepSize[i];
      if ((NumIntervals*Factor) >= (max-min))
         then OKAY:=true
         else i:=succ(i);
   until (OKAY OR (i = 10));
   min:=min*Scale;
   max:=max*Scale;
   if (min >= 0)
      then begin
         PosIntervals:=NumIntervals;
         NegIntervals:=0;
         min:=0;
         max:=NumIntervals*Factor*Scale;
      end   (then)
   else if (max <= 0)
      then begin
         PosIntervals:=0;
```

```
            NegIntervals:=NumIntervals;
            min:=-NumIntervals*Factor*Scale;
            max:=0;
         end   {else if}
      else   {min < 0 < max}
         begin
            PosIntervals:=trunc(max/(max-min)*NumIntervals+1);
            NegIntervals:=trunc(min/(min-max)*NumIntervals+1);
            NumIntervals:=PosIntervals+NegIntervals;
            min:=-NegIntervals*Factor*Scale;
            max:= PosIntervals*Factor*Scale;
         end;   {else}
   end;   {MinMaxLin}

procedure MinMaxLog ( VAR min        : real;
                      VAR max        : real;
                      VAR Scale      : real;
                      VAR NumDecades : byte
                    );

   begin   {MinMaxLog}
      if min > 0 then if min < 1
         then min:=exp10(pred(trunc(log10(min))))
         else min:=exp10     (trunc(log10(min)));
      if max > 0 then if max < 1
         then max:=exp10     (trunc(log10(max)))
         else max:=exp10(succ(trunc(log10(max))));
      if (min = 0) then min:=max*exp10(-MaxDecades);
      NumDecades:=round(log10(max)-log10(min));
      if NumDecades > MaxDecades then begin
         min:=exp10(round(log10(max)-MaxDecades));
         NumDecades:=MaxDecades;
      end;   {if}
      Scale:=min;
   end;   {MinMaxLog}

procedure Skip (     d       : integer;
                     s       : integer;
                     i       : integer;
                VAR SkipNum : SkipType
               );

   begin   {Skip}
      if ((d<=s/5) AND (i>=4))
         then SkipNum:=5
      else if ((d<=s/4) AND (i>=6))
         then SkipNum:=4
      else if ((d<=s/3) AND (i>=8))
         then SkipNum:=3
      else if ((d<=s/2) AND (i>=10))
         then SkipNum:=2
      else SkipNum:=1;
   end;   {Skip}

procedure LinearAxis ( axis         : char;
                       min          : real;
                       max          : real;
                       Scale        : real;
                       Factor       : integer;
                       NumIntervals : byte;
                       PosIntervals : byte;
                       NegIntervals : byte;
                       units        : string
                     );
   var
      i            : integer;
```

```
    j             : integer;
    Plot1         : integer;
    Plot2         : integer;
    OutScale      : integer;
    OutX          : integer;
    OutY          : integer;
    OutFactor     : WrkString;
    XAXIS         : boolean;
    z1            : integer;
    z2            : integer;
    z3            : integer;
    z4            : integer;
    for_limit     : integer;
    OK_to_Draw    : boolean;
    outstr        : string [20];

begin   (LinearAxis)
   XAXIS:=(UpCase(axis) = 'I');
   with world do begin
      if XAXIS
         then begin
            z1:=sx_1; z2:=sx_2; z3:=sy_1; z4:=sy_2;
         end
         else begin
            z1:=sy_1; z2:=sy_2; z3:=sx_1; z4:=sx_2;
         end;
   end;   (with)
   if XAXIS
      then begin
         Plot1:=WhereX (0);
         Line (Plot1,z3,Plot1,z4);
         Line (z1,z3,z2,z3);
         Line (z1,z4,z2,z4);
         PlotX:=Plot1;
         PlotY:=z3+char_height;
         ZeroX:=Plot1;
      end
      else begin
         Plot1:=WhereY (0);
         Line (z3,Plot1,z4,Plot1);
         Line (z3,z1,z3,z2);
         Line (z4,z1,z4,z2);
         PlotX:=z3-4*char_width;
         PlotY:=Plot1-(char_height div 2);
         ZeroY:=Plot1;
      end;
   OutTextXY (PlotX,PlotY,'0');
   for j:=1 to 2 do begin
      for_limit:=PosIntervals+(j-1)*(NegIntervals-PosIntervals);
      for i:=1 to for_limit do begin
         Plot2:=Plot1+(3-2*j)*round(i*(z2-z1)/NumIntervals);
         Str ((3-2*j)*i*Factor,OutFactor);
         if XAXIS
            then begin
               Line (Plot2,z3,Plot2,z3-MajorTic);
               Line (Plot2,z4,Plot2,z4+MajorTic);
               PlotX:=Plot2-(Length(OutFactor) div 2)*char_width;
               PlotY:=z3+char_height;
               OK_to_Draw:=(i mod SkipX = 0);
            end   (then)
            else begin
               Line (z3,Plot2,z3+MajorTic,Plot2);
               Line (z4,Plot2,z4-MajorTic,Plot2);
               PlotX:=z3-(3+j)*char_width;
               PlotY:=Plot2-(char_height div 2);
               OK_to_Draw:=(i mod SkipY = 0);
```

```
                end;
             if OK_to_Draw then OutTextXY (PlotX,PlotY,OutFactor);
          end;   (for i)
       end;   (for j)
       if XAXIS
          then begin
             OutX:=((z1+z2) div 2)-4*char_width;
             OutY:=z3+3*char_height;
          end   (then)
          else begin
             OutX:=z3-(10+MaxDecades)*char_width;
             OutY:=(z1+z2-char_height) div 2;
          end;   (else)
       if (Scale <> 0) then begin
          OutScale:=round(log10(Scale));
          Str (OutScale,OutFactor);
          if OutFactor <> '0' then begin
             PlotX:=OutX;
             PlotY:=OutY;
             outstr:='(x10   )';
             OutTextXY (PlotX,PlotY,outstr);
             PlotX:=OutX;
             PlotY:=OutY-(char_height div 2);
             outstr:='    '+OutFactor;
             OutTextXY (PlotX,PlotY,outstr);
          end;   (if)
       end;   (then)
    end;   (LinearAxis)

procedure LogAxis ( axis       : char;
                    min        : real;
                    max        : real;
                    Scale      : real;
                    NumDecades : byte;
                    units      : string
                  );

   var
      i         : integer;
      j         : integer;
      k         : integer;
      Plot1     : integer;
      Plot2     : integer;
      NewPlot   : integer;
      OutX      : integer;
      OutY      : integer;
      OutScale  : integer;
      OutFactor : Wrkstring;
      z1        : integer;
      z2        : integer;
      z3        : integer;
      z4        : integer;
      XAXIS     : boolean;

   begin   (LogAxis)
      XAXIS:=(UpCase(axis) = 'I');
      with world do begin
         if XAXIS
            then begin
               z1:=sx_1; z2:=sx_2; z3:=sy_1; z4:=sy_2;
            end
            else begin
               z1:=sy_1; z2:=sy_2; z3:=sx_1; z4:=sx_2;
            end;
      end;   (with)
      if XAXIS
```

```
            then begin
               Line (z1,z3,z2,z3);
               Line (z1,z4,z2,z4);
            end
            else begin
               Line (z3,z1,z3,z2);
               Line (z4,z1,z4,z2);
            end;
         Plot1:=z1;
         for i:=1 to NumDecades+1 do begin
            NewPlot:=round(z1+i*(z2-z1)/NumDecades);
            for k:=1 to i do OutFactor[k]:=power[k];
            OutFactor[0]:=char(i);
            if XAXIS
               then begin
                  Line (Plot1,z3,Plot1,z3-MajorTic);
                  Line (Plot1,z4,Plot1,z4+MajorTic);
                  PlotX:=Plot1-(i*char_width) div 2;
                  PlotY:=z3+char_height;
               end
               else begin
                  Line (z3,Plot1,z3+MajorTic,Plot1);
                  Line (z4,Plot1,z4-MajorTic,Plot1);
                  PlotX:=z3-(MaxDecades+1)*char_width;
                  PlotY:=Plot1;
               end;
            OutTextXY (PlotX,PlotY,OutFactor);
            if i <= NumDecades then for j:=2 to 9 do begin
               Plot2:=round(Plot1+(NewPlot-Plot1)*LogTable[j]);
               if XAXIS
                  then begin
                     Line (Plot2,z3,Plot2,z3-MinorTic);
                     Line (Plot2,z4,Plot2,z4+MinorTic);
                  end
                  else begin
                     Line (z3,Plot2,z3+MinorTic,Plot2);
                     Line (z4,Plot2,z4-MinorTic,Plot2);
                  end;
            end;   (for j)
            Plot1:=NewPlot;
         end;   (for i)
         if XAXIS
            then begin
               OutX:=(z1+z2) div 2 - 4*char_width;
               OutY:=z3+3*char_height;
            end
            else begin
               OutX:=z3-(10+MaxDecades)*char_width;
               OutY:=(z1+z2+char_height) div 2;
            end;
         if (scale <> 0) then begin
            OutScale:=round(log10(Scale));
            Str (OutScale,OutFactor);
            if OutFactor <> '0' then begin
               PlotX:=OutX;
               PlotY:=OutY;
               OutTextXY (PlotX,PlotY,'(x10    )');
               PlotX:=OutX;
               PlotY:=OutY-(char_height div 2);
               OutFactor:='     '+OutFactor;
               OutTextXY (PlotX,PlotY,OutFactor);
            end;   (if)
         end;   (then)
      end;   (LogAxis)

procedure LinearScale ( VAR z      : PlotXYPtr;
```

```
                        VAR min   : real;
                        VAR max   : real;
                            Scale : real;
                            N     : integer
                      );

   var i : integer;

   begin   {LinearScale}
      for i:=1 to N do
         z^[i]:=z^[i]/Scale;
      min:=min/Scale;
      max:=max/Scale;
   end;   {LinearScale}

procedure LogScale ( VAR z     : PlotXYPtr;
                     VAR min   : real;
                     VAR max   : real;
                         Scale : real;
                         N     : integer
                   );

   var i : integer;

   begin   {LogScale}
      for i:=1 to N do begin
         if (z^[i] < min) then z^[i]:=min;
         z^[i]:=log10(z^[i]/Scale);
      end;   {for}
      min:=log10(min/Scale);
      max:=log10(max/Scale);
   end;   {LogScale}

procedure GraphArray (x : PlotXYPtr;
                      y : PlotXYPtr;
                      N : integer);

   var
      new_x : integer;
      new_y : integer;
      old_x : integer;
      old_y : integer;
      i     : integer;

   begin   {GraphArray}
      with world do begin
         new_x:=WhereX (x^[1]);
         new_y:=WhereY (y^[1]);
         for i:=2 to N do begin
            old_x:=new_x;
            old_y:=new_y;
            new_x:=WhereX (x^[i]);
            new_y:=WhereY (y^[i]);
            Line (old_x,old_y,new_x,new_y);
         end;   {for}
      end;   {with}
   end;   {GraphArray}

procedure DrawGraph (     x             : PlotXYPtr; (* x data to graph     *)
                          y             : PlotXYPtr; (* y data to graph     *)
                          NumPlotPoints : integer;   (* num of points in A  *)
                          Plot          : byte;      (* type of plot        *)
                          left_x        : integer;   (* left window edge    *)
                          top_y         : integer;   (* top window edge     *)
                          right_x       : integer;   (* right window edge   *)
                          bottom_y      : integer;   (* bottom window edge  *)
```

```
                    VAR minx          : real;
                    VAR maxx          : real;
                    VAR miny          : real;
                    VAR maxy          : real;
                        horiz_units   : string;
                        vert_units    : string;
                    VAR error         : byte
                   );


begin   {DrawGraph}
   error:=0;
   SetTextStyle (SmallFont,HorizDir,char_size);
   char_height:=TextHeight ('H');
   char_width :=TextWidth  ('W');
   left_x:=left_x+(11+MaxDecades)*char_width;
   bottom_y:=bottom_y-3*char_height;
   XLinear:=((Plot = Linear) OR (Plot = LinLog));
   YLinear:=((Plot = Linear) OR (Plot = LogLin));
   Drop (x,y,Plot,start);
   for i:=start to NumPlotPoints do begin
      x^[i-start+1]:=x^[i];
      y^[i-start+1]:=y^[i];
   end;   {for}
   NumPlotPoints:=NumPlotPoints-start+1;
   minx:=x^[1];
   maxx:=x^[1];
   miny:=y^[1];
   maxy:=y^[1];
   for i:=2 to NumPlotPoints do begin
      if minx > x^[i]
         then minx:=x^[i]
      else if maxx < x^[i]
         then maxx:=x^[i];
      if miny > y^[i]
         then miny:=y^[i]
      else if maxy < y^[i]
         then maxy:=y^[i];
   end;   {for}
   if (NOT XLinear) AND (minx <= 0) then error:=1;
   if (NOT YLinear) AND (miny <= 0) then error:=1;
   if (error = 0) then begin
      if XLinear
         then MinMaxLin (minx,maxx,ScaleX,FactorX,NumIntervalsX,
                         PosIntervalsX,NegIntervalsX)
         else MinMaxLog (minx,maxx,ScaleX,NumDecadesX);
      if YLinear
         then MinMaxLin (miny,maxy,ScaleY,FactorY,NumIntervalsY,
                         PosIntervalsY,NegIntervalsY)
         else MinMaxLog (miny,maxy,ScaleY,NumDecadesY);
      SetUpWorld (minx,miny,maxx,maxy,left_x,bottom_y,right_x,top_y);
      if XLinear
         then begin
            Skip        ((right_x-left_x),GetMaxX,NumIntervalsX,SkipX);
            LinearAxis  ('i',minx,maxx,ScaleX,FactorX,NumIntervalsX,
                         PosIntervalsX,NegIntervalsX,horiz_units);
            LinearScale (x,minx,maxx,ScaleX,NumPlotPoints);
         end   {then}
         else begin
            LogAxis   ('i',minx,maxx,ScaleX,NumDecadesX,horiz_units);
            LogScale  (x,minx,maxx,ScaleX,NumPlotPoints);
         end;   {then}
      if YLinear
         then begin
            Skip        ((bottom_y-top_y),GetMaxY,NumIntervalsY,SkipY);
            LinearAxis  ('d',miny,maxy,ScaleY FactorY,NumIntervalsY,
```

```
                              PosIntervalsY,NegIntervalsY,vert_units);
                  LinearScale (y,miny,maxy,ScaleY,NumPlotPoints);
               end   {then}
               else begin
                  LogAxis   ('d',miny,maxy,ScaleY,NumDecadesY,vert_units);
                  LogScale  (y,miny,maxy,ScaleY,NumPlotPoints);
               end;   {else}
            with world do
               SetUpWorld (minx,miny,maxx,maxy,sx_1,sy_1,sx_2,sy_2);
            GraphArray (x,y,NumPlotPoints);
         end;   {if error = 0}
      end;   {DrawGraph}

(*****************************************************************************)

var
  GraphDriver, GraphMode, Error : integer;


procedure CgaDriverProc; external;
{$L \pascal4\graphix\CGA.OBJ }

procedure EgaVgaDriverProc; external;
{$L \pascal4\graphix\EGAVGA.OBJ }

procedure HercDriverProc; external;
{$L \pascal4\graphix\HERC.OBJ }

procedure SmallFontProc; external;
{$L c:\pascal4\graphix\LITT.OBJ }

procedure TriplexFontProc; external;
{$L c:\pascal4\graphix\TRIP.OBJ }


procedure Abort(Msg : string);
begin
  Writeln(Msg, ': ', GraphErrorMsg(GraphResult));
  Halt(1);
end;

begin   {Initialization section}

  { Register all the drivers }
  if RegisterBGIdriver(@CGADriverProc) < 0 then
    Abort('CGA');
  if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
    Abort('EGA/VGA');
  if RegisterBGIdriver(@HercDriverProc) < 0 then
    Abort('Herc');

  { Register all the fonts }
  if RegisterBGIfont(@SmallFontProc) < 0 then
    Abort('Small');

  GraphDriver := Detect;                  { autodetect the hardware }
  InitGraph(GraphDriver, GraphMode, '');  { activate graphics }
  if GraphResult <> grOk then begin       { any errors? }
     Writeln('Graphics initialization error: ', GraphErrorMsg(GraphDriver));
     Halt(1);
  end;
  CloseGraph;
end.   {Unit DrawGraf}
```

A-3.10 GraphText

Unit GraphText contains two routines:

1. Text_To_Cart; and

2. PrintText.

The first procedure converts text coordinates to screen coordinates in the current graphics system. The second procedure replaces procedure PrintText in unit GlobalProcedures for writing text to the screen in graphics mode.

APPENDIX A

```
unit GraphText;

interface
uses
   Crt,
   Graph;

procedure Text_To_Cart (     tx,ty : integer;
                         VAR cx,cy : integer);

procedure PrintText (s:string; x,y:integer);

(*************************************************************************)

implementation


procedure Text_To_Cart (     tx,ty : integer;
                         VAR cx,cy : integer);

   begin   (Text_To_Cart)
      cx:=(tx-1)*((GetMaxX+1) div 80);
      cy:=(ty-1)*((GetMaxY+1) div 25);
   end;   (Text_To_Cart)


procedure PrintText (s : string;
                     x : integer;
                     y : integer);

   const
      NormSize = 1;

   begin   (PrintText)
      GotoXY (x + length(s),y);
      SetTextStyle (DefaultFont,HorizDir,NormSize);
      SetTextJustify (LeftText,TopText);
      Text_To_Cart (x,y,x,y);
      MoveTo (x,y);
      OutText (s);
      MoveTo (x+TextWidth(s),y);
   end;   (PrintText)


(*************************************************************************)

begin   (Initialization)
end.   (unit GraphText)
```

A-3.11 Async

Unit Async contains routines which allow Turbo Pascal to talk to an asynchronous port (COM1 or COM2). The routines were developed from Technical Information Sheet number 226, dated August 1, 1986, from Borland International, Inc. The following routines are present:

1. Async_Init--initializes asynchronous routines.

2. Async_Open--opens the COM port; sets up the interrupt vector and data buffer; sets the baud rate, parity, word size, and stop bits.

3. Async_Buffer_Check--reads a character from the buffer if present.

4. Async_Send--transmits a character.

5. Async_Send_String--transmits a string of characters.

6. Async_Close--turns off COM port interrupts.

7. Async_Change--changes COM port parameters (baud rate, parity, word size, and stop bits).

APPENDIX A

```
(*****************************************************************************)
(***                                                                       ***)
(***  ASYNC.INC has been developed from Technical Information Number 226   ***)
(***  from Borland International, dated August 1, 1986. The program is    ***)
(***  highly dependent on the IBM PC and PC-DOS 2.Ø.                       ***)
(***                                                                       ***)
(*****************************************************************************)
(***                                                                       ***)
(***  ASYNC.INC - Asynchronous Communications Routines.                    ***)
(***                                                                       ***)
(***  Entry points:                                                        ***)
(***     Async_Init                                                        ***)
(***        Performs initialization.                                       ***)
(***                                                                       ***)
(***     Async_Open (Port, Baud : integer;                                 ***)
(***                 Parity : char;                                        ***)
(***                 WordSize,StopBits : integer) : boolean;               ***)
(***        Sets up interrupt vector, initialize the COM port for          ***)
(***        processing, sets pointers to the buffer. Returns FALSE if      ***)
(***        COM port not installed.                                        ***)
(***                                                                       ***)
(***     Async_Buffer_Check (var C : char) : boolean                       ***)
(***        If a character is available, returns TRUE and moves the        ***)
(***        character from the buffer to the parameter; otherwise,         ***)
(***        returns FALSE.                                                 ***)
(***                                                                       ***)
(***     Async_Send (C : char);                                            ***)
(***        Transmits the character.                                       ***)
(***                                                                       ***)
(***     Async_Send_String (S : string)                                    ***)
(***        Calls Async_Send to send each character of S.                  ***)
(***                                                                       ***)
(***     Async_Close                                                       ***)
(***        Turn off the COM port interrupts. This must be called before   ***)
(***        exiting the progra, otherwise unpredictable results will       ***)
(***        occur and it will be necessary to reboot.                      ***)
(***                                                                       ***)
(*****************************************************************************)

unit Async;

interface

uses DOS;

type
   real = extended;

procedure Async_Init;

procedure Async_Close;

function Async_Open ( ComPort  : integer;
                      BaudRate : integer;
                      Parity   : char;
                      WordSize : integer;
                      StopBits : integer ) : boolean;

function Async_Buffer_Check (var C:char) : boolean;

procedure Async_Send (C:char);

procedure Async_Send_String (S:string);

procedure Async_Change ( BaudRate : integer;
                         Parity   : char;
```

```
                        WordSize : integer;
                        StopBits : integer);

(*****************************************************************************)

implementation

const
   UART_THR = $00;      (* offset from base of UART registers for IBM PC.   *)
   UART_RBR = $00;
   UART_IER = $01;
   UART_IIR = $02;
   UART_LCR = $03;
   UART_MCR = $04;
   UART_LSR = $05;
   UART_MSR = $06;

   I8088_IMR = $21;     (* port address of the interrupt Mask Register.     *)

   Async_Dseg_Save : integer = 0;      (* Save DS reg in Code Segment       *)
                                       (* for interrupt routine.            *)

   Async_Buffer_Max = 4095;

var
   Async_Buffer          : array [0..Async_Buffer_Max] of char;
   Async_Open_Flag       : boolean;
   Async_Port            : integer;    (* Current open port number (1 or 2) *)
   Async_Base            : integer;    (* Base for current open port.       *)
   Async_IRQ             : integer;    (* IRQ for current open port.        *)
   Async_Buffer_Overflow : boolean;    (* Buffer overflow?                  *)
   Async_Buffer_Used     : integer;
   Async_MaxBufferUsed   : integer;

      (*** Async_Buffer is empty if Head = Tail ***)
   Async_Buffer_Head     : integer;    (* Loc'n in buffer to put next char. *)
   Async_Buffer_Tail     : integer;    (* Loc'n in buffer to get next char. *)
   Async_Buffer_NewTail  : integer;

   Async_BIOS_Port_Table : array [1..2] of integer absolute $40:0;
      (*** This table is initialized by BIOS equipment determination code ***)
      (*** at boot time to contain the base addresses for the installed   ***)
      (*** async adapters. A value of 0 means "not installed."            ***)

const
   Async_Num_Bauds = 8;
   Async_Baud_Table : array [1..Async_Num_Bauds] of record
                                                      Baud : integer;
                                                      Bits : integer;
                                                   end   (record)
                    = ((Baud:110;  Bits:$00),
                       (Baud:150;  Bits:$20),
                       (Baud:300;  Bits:$40),
                       (Baud:600;  Bits:$60),
                       (Baud:1200; Bits:$80),
                       (Baud:2400; Bits:$A0),
                       (Baud:4800; Bits:$C0),
                       (Baud:9600; Bits:$E0));

procedure BIOS_RS232_Init (ComPort,ComParm : integer);
   (*** Issue Interrupt $14 to initialize the UART. See the IBM Technical ***)
   (*** Reference Manual for the format of ComParm.                        ***)

   var
      Regs : registers;
```

```
   begin   {BIOS_RS232_Init}
      with Regs do begin
         ax:=ComParm and $00FF;        (* AH:=0; AL:=ComParm              *)
         dx:=ComPort;
         Intr ($14,Regs);
      end;   {with}
   end;   {BIOS_RS232_Init}

procedure DOS_Set_Interrupt (v,s,o:integer);
   (*** call DOS to set an interrupt vector                               *)

   var
      Regs : Registers;

   begin   {DOS_Set_Interrupt}
      with Regs do begin
         ax:=$2500 + (v and $00FF);
         ds:=s;
         dx:=o;
         MsDos (Regs);
      end;   {with}
   end;   {DOS_Set_Interrupt}

(*****************************************************************************)
(***                                                                       ***)
(***  ASYNCISR.INC - Interrupt Service Routine                             ***)
(***                                                                       ***)
(*****************************************************************************)

procedure Async_ISR;
   (*** Interrupt Service Routine. Invoked when the UART has received a   ***)
   (*** byte of data from the communication line. Written in machine      ***)
   (*** language, the assembly code is shown as comments.                 ***)

   begin   {Async_ISR}
      (*** Note: on entry, Turbo Pascal has already PUSHed BP and SP.     ***)
      Inline (                        (* Save all registers used.         *)
         $50/                         (* PUSH AX                          *)
         $53/                         (* PUSH BX                          *)
         $52/                         (* PUSH DX                          *)
         $1E/                         (* PUSH DS                          *)
         $FB/                         (* STI                              *)

         (* Set up the DS register to point to Turbo Pascal's data segment. *)
         $2E/$FF/$36/Async_Dseg_Save/  (* PUSH CS:Async_Dseg_Save          *)
         $1F/                         (* POP DS                           *)

         (* Get the incoming character.                                     *)
         (* Async_Buffer[Async_Buffer_Head]:=Chr(Port[UART_RBR+Async_Base]) *)
         $8B/$16/Async_Base/          (* MOV DX,Async_Base                 *)
         $EC/                         (* IN AL,DX                         *)
         $8B/$1E/Async_Buffer_Head/   (* MOV BX,Async_Buffer_Head         *)
         $88/$87/Async_Buffer/        (* MOV Async_Buffer [BX],AL         *)

         (* Async_Buffer_NewHead := Async_Buffer_Head + 1                   *)
         $43/                         (* INC BX                           *)

         (* if Async_Buffer_NewHead > Async_Buffer_Max                      *)
         (*    then Async_Buffer_NewHead := 0                              *)
         $81/$FB/Async_Buffer_Max/    (* CMP BX,Async_Buffer_Max          *)
         $7E/$02/                     (* JLE L001                         *)
         $33/$DB/                     (* XOR BX,BX                        *)

         (* if Async_Buffer_NewHead = Async_Buffer_Tail                     *)
         (*    then Async_Buffer_Overflow := true                          *)
         (*    else begin                                                  *)
```

```
            (*       Async_Buffer_Head := Async_Buffer_NewHead;                *)
            (*       Async_Buffer_Used := Async_Buffer_Used + 1;               *)
            (*       if Async_Buffer_Used > Async_MaxBufferUsed then           *)
            (*          Async_MaxBufferUsed := Async_Buffer_Used;              *)
            (*    end;                                                        *)

            (* L001: then                                                     *)
            $3B/$1E/Async_Buffer_Tail/    (* CMP BX, Async_Buffer_Tail       *)
            $75/$08/                      (* JNE L002                        *)
            $C6/$06/Async_Buffer_Overflow/$01/  (* MOV Async_Buffer_Overflow,1 *)
            $90/                          (* NOP generated by assembler      *)
            $EB/$16/                      (* JMP SHORT L003                  *)

            (* L002: else                                                     *)
            $89/$1E/Async_Buffer_Head/    (* MOV Async_Buffer_Head,BX        *)
            $FF/$06/Async_Buffer_Used/    (* INC Async_Buffer_Used           *)
            $8B/$1E/Async_Buffer_Used/    (* MOV BX,Async_Buffer_Used        *)
            $3B/$1E/Async_MaxBufferUsed/  (* CMP BX,Async_MaxBufferUsed      *)
            $7E/$04/                      (* JLE L003                        *)
            $89/$1E/Async_MaxBufferUsed/  (* MOV Async_MaxBufferUsed,BX      *)

            (* L003: end if                                                   *)
            (* disable interrupts                                           *)
            $FA/                          (* CLI                             *)
            (* Port [$20] := $20;            use non-specific EOI            *)
            $B0/$20/                      (* MOV AL,$20                      *)
            $E6/$20/                      (* OUT $20,AL                      *)

            (* Restore the last registers then use IRET to return. The last  *)
            (* two POPs are required because Turbo Pascal PUSHes these regs  *)
            (* before we get control, although the manual doesn't show it.   *)
            $1F/                          (* POP DS                          *)
            $5A/                          (* POP DX                          *)
            $5B/                          (* POP BX                          *)
            $58/                          (* POP AX                          *)
            $5C/                          (* POP SP                          *)
            $5D/                          (* POP BP                          *)
            $CF);                         (* IRET                            *)
      end;   {Async_ISR}

   procedure Async_Init;
      (*** Initialize variables.                                            ***)

      begin   {Async_Init}
         Async_DSeg_Save       := DSeg;
         Async_Open_Flag       := FALSE;
         Async_Buffer_Overflow := FALSE;
         Async_Buffer_Used     := 0;
         Async_MaxBufferUsed   := 0;
      end;   {Async_Init}

   procedure Async_Close;
      (*** Reset the interrupt system when UART interrupts no longer needed. ***)

      var
         i,m : integer;

      begin
         if Async_Open_Flag then begin
            (* Disable the IRQ on the 8259.                                  *)
            Inline ($FA);                 (* Disable the interrupts.         *)
            i:= Port [I8088_IMR];         (* Get the Interrupt Mask Register *)
            m:= 1 shl Async_IRQ;          (* Set mask to turn off interrupt  *)
            Port [I8088_IMR] := i or m;

            (* disable the 8250 data ready interrupt                         *)
```

```
        Port [UART_IER + Async_Base] := 0;

        (* disable OUT2 on the 8250                                    *)
        Port [UART_MCR + Async_Base] := 0;
        Inline ($FB);                  (* enable interrupts            *)

        (* Re-initialize data areas to know the port is closed.        *)
        Async_Open_Flag:=FALSE;
      end;   (if)
   end;   (Async_Close)

function Async_Open ( ComPort  : integer;
                      BaudRate : integer;
                      Parity   : char;
                      WordSize : integer;
                      StopBits : integer ) : boolean;
   (*** Open a communication port.                                   ***)

   var
      ComParm : integer;
      i,m     : integer;

   begin   (Async_Open)
      if Async_Open_Flag then Async_Close;
      if (ComPort=2) AND (Async_BIOS_Port_Table[2] <> 0)
         then Async_Port:=2
         else Async_Port:=1;           (* Default to COM1:            *)
      Async_Base:=Async_BIOS_Port_Table[Async_Port];
      Async_IRQ:=Hi(Async_Base)+1;
      if (Port [UART_IIR+Async_Base] and $00F8) <> 0
         then Async_Open:=FALSE
         else begin
            Async_Buffer_Head:=0;
            Async_Buffer_Tail:=0;
            Async_Buffer_Overflow:=FALSE;

            (*** Build the ComParm for RS232_Init. See the Technica'  ***)
            (*** Reference Manual for description.                    ***)
            ComParm:=$0000;

            (*** Set up the bits for the baud rate.                   ***)
            i:=0;
            repeat
               i:=succ(i);
            until (Async_Baud_Table[i].Baud = BaudRate)
                  OR (i = Async_Num_Bauds);
            ComParm:=ComParm or Async_Baud_Table[i].Bits;
            if parity in ['E','e']
               then ComParm := ComParm OR $0018
               else if parity in ['O','o']
                  then ComParm := ComParm OR $0008
                  else ComParm := ComParm OR $0000; (* Default to no parity *)
            if WordSize = 7
               then ComParm:=ComParm OR $0002
               else ComParm:=ComParm OR $0003;   (* Default to 8 data bits. *)
            if StopBits = 2
               then ComParm:=ComParm OR $0004
               else ComParm:=ComParm or $0000;   (* Default to 1 stop bit.  *)

            (*** use the BIOS COM port initialization routine to save ***)
            (*** typing the code.                                     ***)
            BIOS_RS232_Init (Async_Port-1,ComParm);
            DOS_Set_Interrupt (Async_IRQ+8,CSeg,Ofs(Async_ISR));

            (*** Read the RBR and reset any possible pending error    ***)
            (*** conditions. First turn off the Divisor Access Latch Bit ***)
```

```
            (*** to allow access to RBR, etc.                              ***)
            Inline ($FA);              (* Disable interrupts.               *)

            Port [UART_LCR+Async_Base]:=Port [UART_LCR+Async_Base] AND $7F;
            (*** Read the Line Status Register to reset any error it       ***)
            (*** indicates                                                 ***)
            i:=Port [UART_LSR+Async_Base];
            (*** Read the Receiver Buffer Register in case it contains     ***)
            (*** a character.                                              ***)
            i:=Port [UART_RBR+Async_Base];

            (*** Enable the IRQ on the 8259 controller.                    ***)
            i:=Port [I8088_IMR];       (* Get the Interrupt Mask Register.  *)
            m:=(1 shl Async_IRQ) XOR $00FF;
            Port [I8088_IMR]:=i AND m;

            (*** Enable the Data Ready Interrupt on the 8250.              ***)
            Port [UART_IER+Async_Base]:=$01;

            (*** Enable OUT2 on 8250.                                      ***)
            i:=Port [UART_MCR+Async_Base];
            Port [UART_MCR+Async_Base]:=i OR $08;

            Inline ($FB);              (* enable interrupts                 ***)
            Async_Open_Flag:=TRUE;
            Async_Open:=TRUE;
         end;   {if-else}
   end;   {Async_Open}

function Async_Buffer_Check (var C:char) : boolean;
(*** See if a character has been received; return it if yes.               ***)

   begin   {Async_Buffer_Check}
      if Async_Buffer_Head = Async_Buffer_Tail
         then Async_Buffer_Check:=FALSE
         else begin
            C:=Async_Buffer[Async_Buffer_Tail];
            Async_Buffer_Tail:=succ (Async_Buffer_Tail);
            if Async_Buffer_Tail > Async_Buffer_Max then Async_Buffer_Tail:=0;
            Async_Buffer_Used:=Async_Buffer_Used-1;
            Async_Buffer_Check:=TRUE;
         end;   {else}
   end;   {Async_Buffer_Check}

procedure Async_Send (C:char);
   (*** Transmits a character.                                             ***)

   var
      i,m,counter : integer;

   begin   {Async_Send}
      Port [UART_MCR+Async_Base]:=$0B; (* Turn on OUT2, DTR, and RTS.       *)

      (*** Wait for CTS                                                    ***)
      counter:=MaxInt;
      while (counter <> 0) AND ((Port [UART_MSR+Async_Base] AND $10) = 0) do
         counter:=pred (counter);

      (*** Wait for Transmit Hold Register Empty (THRE).                   ***)
      if counter <> 0 then counter:=MaxInt;
      while (counter <> 0) AND ((Port [UART_LSR+Async_Base] AND $20) = 0) do
         counter:=pred (counter);
      if counter <> 0
         then begin
            (*** Send the character.                                       ***)
            Inline ($FA);              (* disable interrupts                *)
```

```
            Port [UART_THR+Async_Base]:=Ord(C);
            Inline ($FB);                    (* enable interrupts                *)
         end   (then)
         else
            writeln ('<<<TIMEOUT>>>');
   end;   (Async_Send)

procedure Async_Send_String (S:string);
   (*** Transmit a string.                                                       *)

   var
      i : integer;

   begin   (Async_Send_String)
      for i:=1 to length (S) do
         Async_Send (S[i]);
   end;   (Async_Send_String)

procedure Async_Change ( BaudRate : integer;
                         Parity   : char;
                         WordSize : integer;
                         StopBits : integer);
   (*** Changes communications parameters. The BIOS routines cannot be     ***)
   (*** used because they drop DTR.                                         ***)

   const
      Num_Bauds = 15;
      divisor_table : array [1..Num_Bauds] of record
                                                 baud    : integer;
                                                 divisor : integer;
                                              end   (record)
                    = ((baud:50;   divisor:2304),
                       (baud:75;   divisor:1536),
                       (baud:110;  divisor:1047),
                       (baud:134;  divisor:857),
                       (baud:150;  divisor:768),
                       (baud:300;  divisor:384),
                       (baud:600;  divisor:192),
                       (baud:1200; divisor:96),
                       (baud:1800; divisor:64),
                       (baud:2000; divisor:58),
                       (baud:2400; divisor:48),
                       (baud:3600; divisor:32),
                       (baud:4800; divisor:24),
                       (baud:7200; divisor:16),
                       (baud:9600; divisor:12));

   var
      i   : integer;
      dv  : integer;
      lcr : integer;

   begin   (Async_Change)
      (*** Build the Line Control Register and find the divisor          ***)
      (*** (for the baud rate).                                          ***)
      i:=0;
      repeat
         i:=succ (i);
      until (Divisor_Table[i].Baud = BaudRate) OR (i = Num_Bauds);
      dv:=Divisor_Table[i].divisor;

      lcr:=0;
      case Parity of
         'E' : lcr:=lcr OR $18;        (* even  parity                      *)
         'O' : lcr:=lcr OR $08;        (* odd   parity                      *)
         'N' : lcr:=lcr OR $00;        (* no    parity                      *)
```

```
            'M' : lcr:=lcr OR $28;          (* Mark  parity                    *)
            'S' : lcr:=lcr OR $38;          (* Space parity                    *)
             else lcr:=lcr OR $00;          (* Default to no parity.           *)
         end;   (case)

         case WordSize of
            5:   lcr:=lcr OR $00;
            6:   lcr:=lcr OR $01;
            7:   lcr:=lcr OR $02;
            8:   lcr:=lcr OR $03;
            else lcr:=lcr OR $04;          (* default to 8 bits.              *)
         end;   (case)

         if StopBits = 2
            then lcr:=lcr OR $04
            else lcr:=lcr OR $00;          (* default to 1 stop bit.          *)
         lcr:=lcr AND $7F;                 (* make certain the DLAB is off.   *)
         Inline ($FA);                     (* disable interrupts.             *)

         (*** turn on DLAB to access the divisor.                            ***)
         Port [UART_LCR+Async_Base]:=Port [UART_LCR+Async_Base] OR $80;

         (*** set the divisor.                                               ***)
         Port [Async_Base]:=Lo (dv);
         Port [Async_Base+1]:=Hi (dv);

         (*** turn off the DLAB and set the new communications parameters.   ***)
         Port [UART_LCR+Async_Base]:=lcr;

         Inline ($FB);                     (* enable interrupts.              *)
      end;   (Async_Change)

   (*************************************************************************)

   begin   (Initialization)
   end.   (Initialization)
```

DISTRIBUTION

ADMINISTRATOR
DEFENSE TECHNICAL INFORMATION CENTER
CAMERON STATION, BUILDING 5
ATTN DTIC-DDA (12 COPIES)
ALEXANDRIA, VA 22304-6145

ASSISTANT TO THE SECRETARY OF DEFENSE
ATOMIC ENERGY
ATTN EXECUTIVE ASSISTANT
WASHINGTON, DC 20301

DIRECTOR
DEFENSE COMMUNICATIONS AGENCY
ATTN CODE B410
ATTN CODE [illegible] 30
WASHINGTON, DC 20305

DIRECTOR
COMMAND CONTROL ENGINEERING CENTER
ATTN C-660
ATTN G-630
WASHINGTON, DC 20305

DIRECTOR
DEFENSE COMMUNICATIONS ENGINEERING CENTER
ATTN CODE R400
ATTN CODE R123, TECH LIB
ATTN CODE R111
1860 WIEHLE AVENUE
RESTON, VA 22090

ASSISTANT CHIEF OF STAFF FOR
INFORMATION MANAGEMENT
COMMAND SYSTEMS INTEGRATION OFFICE
ATTN DAMO-C4Z
THE PENTAGON
WASHINGTON, DC 20301

DIRECTOR
DEFENSE INTELLIGENCE AGENCY
ATTN DB-4C2, D. SPOHN
WASHINGTON, DC 20301

CHAIRMAN
JOINT CHIEFS OF STAFF
ATTN J-3
ATTN C3S
WASHINGTON, DC 20301

NATIONAL COMMUNICATIONS SYSTEM
DEPARTMENT OF DEFENSE
OFFICE OF THE MANAGER
ATTN NCS-TS, D. BODSON
WASHINGTON, DC 20305

DIRECTOR
DEFENSE NUCLEAR AGENCY
ATTN RAEV
ATTN DDST
ATTN RAEE

DIRECTOR
DEFENSE NUCLEAR AGENCY (cont'd)
ATTN TITL
WASHINGTON, DC 20305

OFFICE OF UNDERSECRETARY OF DEFENSE
RESEARCH & ENGINEERING
DMSSO
2 SKYLINE PLACE
SUITE 1403
5203 LEESBURG PIKE
FALLS CHURCH, VA 22041

UNDER SECY OF DEF FOR RSCH & ENGRG
DEPARTMENT OF DEFENSE
ATTN STRATEGIC & SPACE SYS 90SO RM 3E129
ATTN STRAT & THEATER NUC FORCES
WASHINGTON, DC 20301

DEPUTY DIRECTOR FOR THEATRE/TACTICAL C3
SYSTEMS
JOINT STAFF
WASHINGTON, DC 20301

COMMANDER-IN-CHIEF
US FORCES, EUROPE
ATTN ECC3S
APO, NY 09128

ASSISTANT CHIEF OF STAFF FOR
AUTOMATION & COMMUNICATIONS
ATTN DAMO-C4T
ATTN DAMO-C4S
DEPARTMENT OF THE ARMY
WASHINGTON, DC 20360

US ARMY BALLISTIC RESEARCH
LABORATORY
ATTN DRDAR-TSB-S (STINFO)
ABERDEEN PROVING GROUND, MD 21005

US ARMY COMBAT SURVEILLANCE & TARGET
ACQUISITION LABORATORY
ATTN DELET-DD
FT MONMOUTH, NJ 07703

COMMANDER
US ARMY ELECTRONIC SYSTEMS ENGINEERING
INSTALLATION AGENCY
ATTN ASBH-SET-S
FORT HUACHUCA, AZ 85613

US ARMY ENGINEER DIV HUNTSVILLE
DIVSION ENGINEER
ATTN HNDED FD,
PO BOX 1600
HUNTSVILLE, AL 35807

COMMANDER
US ARMY INFORMATION SYSTEMS COMMAND
ATTN CC-OPS-WR, O.P. CONNELL/R. NELSON
FT HUACHUCA, AZ 85613

COMMANDER
US ARMY INFORMATION SYSTEMS
ENGINEERING COMMAND
SUPPORT ACTIVITY
ATTN ASB-TS-A, B. EGBERT
FORT MONMOUTH, NJ 07703-5000

COMMAND
US ARMY MATERIEL COMMAND
ATTN DRCRE
ATTN DRCDE
5001 EISENHOWER AVE
ALEXANDRIA, VA 22333-0001

DIRECTOR
US ARMY MATERIEL SYSTEMS ANALYSIS
ACTIVITY
ATTN DRXSY-MP, LIBRARY
ABERDEEN PROVING GROUND, MD 21005

COMMANDER
US ARMY MISSILE COMMAND
ATTN DRCPM-CF, CHAPARRAL/FAAR
ATTN DRCPM-HD, HELLFIRE/GLD
ATTN DRCPM-PE, PERSHING
ATTN DRCPM-DT, TOW DRAGON
ATTN DRCPM-RS, GENERAL SUPPORT
ROCKET SYS
ATTN DRCPM-HEL, HIGH ENERGY LASER SYS
ATTN DRCPM-ROL, ROLAND
ATTN DRCPM-VI, VIPER
ATTN DRCPM-HA, HAWK
ATTN DRCPM-MP, STINGER
ATTN DRSMI-T, TARGETS MANAGEMENT OFFICE
ATTN DRSMI-U, WEAPONS SYS MGT DIR
ATTN DRSMI-D, PLANS, ANALYSIS,
& EVALUATION
ATTN DRSMI-E, ENGINEERING
ATTN DRSMI-Q, PRODUCT ASSURANCE
ATTN DRMSI-S, MATERIEL MANAGEMENT
ATTN DRSMI-W, MGMT INFO SYSTEMS
REDSTONE ARSENAL, AL 35809

DIRECTOR
US ARMY MISSILE LABORATORY
USAMICOM
ATTN DRSMI-RPR, REDSTONE SCIENTIFIC
INFO CENTER
ATTN DRSMI-RPT, TECHNICAL
INFORMATION DIV
ATTN DRSMI-RN, CHIEF, TECHNOLOGY
INTEGRATION OFFICE
ATTN DRSMI-RA, CHIEF, DARPA PROJECTS
OFFICE

DIRECTOR
US ARMY MISSILE LABORATORY (cont'd)
USAMICOM
ATTN DRSMI-RH, DIR, DIRECTED ENERGY
DIRECTORATE
ATTN DRSMI-RL, SPE ASST GROUND EQUIP &
MISSILE STRUCTURES DIR
ATTN DRSMI-RR, RESEARCH DIR
ATTN DRSMI-RS, SYS ENGR DIR
ATTN DRSMI-RT, TEST & EVAL DIR
ATTN DRSMI-RD, SYST SIMULATION & DEV DIR
ATTN DRSMI-RE, ADVANCED SENSORS DIR
ATTN DRSMI-RK, PROPULSION DIR
ATTN DRSMI-RG, GUIDANCE & CONTROL DIR
REDSTONE ARSENAL, AL 35809

COMMANDER
US ARMY MISSILE & MUNITIONS
CENTER & SCHOOL
ATTN ATSK-CTD-F
REDSTONE ARSENAL, AL 35809

COMMANDER
US ARMY NUCLEAR & CHEMICAL AGENCY
ATTN MONA-WE
7500 BACKLICK ROAD
SPRINGFIELD, VA 22150

OFFICE OF THE ASSIST SEC
OF THE ARMY (RDA)
DEPARTMENT OF THE ARMY
ATTN DAMA-CSS-N
WASHINGTON, DC 20310

CHIEF
US ARMY SATELLITE COMMUNICATIONS
AGENCY
ATTN DRCPM-SC
FT MONMOUTH, NJ 07703

DIRECTOR
TRI/TAC
ATTN TT-E-SS, CHARNICK
FT MONMOUTH, NJ 07703

COMMANDANT
US ARMY WAR COLLEGE
ATTN LIBRARY
CARLISLE BARRACKS, PA 17013

COMMANDER-IN-CHIEF
ATLANTIC
ATTN J6
NORFOLK, VA 23511

COMMANDER
NAVAL ELECTRONIC SYSTEMS COMMAND
ATTN PME 110-241D
WASHINGTON, DC 20360

DISTRIBUTION (cont'd)

NAVAL ELECTRONICS ENGINEERING ACTIVITY,
PACIFIC
BOX 130
ATTN DON O'BRYHIM
PEARL HARBOR, HAWAII 96860-5170

CHIEF OF NAVAL MATERIEL
THEATER NUCLEAR WARFARE PROJECT OFFICE
ATTN PM-23
WASHINGTON, DC 20360

COMMANDER
NAVAL OCEAN SYSTEMS CENTER
ATTN CODE 83, J. STAWISKI
SAN DIEGO, CA 92152

COMMANDING OFFICER
NAVAL ORDNANCE STATION
ATTN STANDARDIZATION DIVISION
INDIAN HEAD, MD 20640

COMMANDER-IN-CHIEF
PACIFIC
ATTN C3S-RF-1
CAMP H. M. SMITH, HI 96861

COMMANDING OFFICER
NAVAL RESEARCH LABORATORY
ATTN CODE 4720, J. DAVIS
WASHINGTON, DC 20375

COMMANDER
NAVAL SURFACE WEAPONS CENTER
ATTN CODE F-56
DAHLGREN, VA 22448

COMMANDER
NAVAL SURFACE WEAPONS CENTER
ATTN CODE F32, E. RATHBURN
ATTN CODE F30
WHITE OAK LABORATORY
SILVER SPRING, MD 20910

DEPARTMENT OF THE NAVY
DIRECTOR, NAVAL TELECOMMUNICATIONS
DIVISION
OFFICE OF THE CHIEF OF NAVAL OPERATIONS
ATTN OP941, HAISLMAIER
ATTN OP943
WASHINGTON, DC 20350

HQ, USAF/SAMI
WASHINGTON, DC 20330

AIR FORCE COMMUNICATIONS COMMAND
ATTN EPPD
SCOTT AFB, IL 62225

COMMANDER
US AIR FORCE SPACE COMMAND
ATTN KRQ
ATTN XPOW
PETERSON AFB, CO 80912

1842 EEG
ATTN EEISG
SCOTT AFB, IL 62225

HEADQUARTERS
ELECTRONIC SYSTEMS DIVISION/YS
ATTN YSEA
HANSCOM AFB, MA 01730

HEADQUARTERS
USAFE
ATTN DCKI
RAMSTEIN AFB, GERMANY

SYSTEM INTEGRATION OFFICE
ATTN SYE
PETERSON AFB, CO 80912

AIR FORCE WEAPONS LABORATORY/DYC
ATTN NTC4, TESD, IESM
KIRTLAND AFB, NM 87117

CENTRAL INTELLIGENCE AGENCY
ATTN OWSR/NED
WASHINGTON, DC 20505

DIRECTOR
FEDERAL EMERGENCY MANAGEMENT AGENCY
NATIONAL PREPAREDNESS PROGRAM SUPPORT
ATTN OFFICE OF RESEARCH
WASHINGTON, DC 20472

DIRECTOR
FEDERAL EMERGENCY MANAGEMENT AGENCY
STATE & LOCAL SUPPORT BR
ATTN SL/EM/SS/LS, LOGISTICS
SUPPORT BRANCH
WASHINGTON, DC 20472

LAWRENCE LIVERMORE NATIONAL LAB
ATTN TECHNICAL INFO DEPT LIBRARY
ATTN L-156, H. CABAYAN, L. MARTIN
PO BOX 808
LIVERMORE, CA 94550

DIRECTOR
NATIONAL SECURITY AGENCY
ATTN R15
9800 SAVAGE ROAD
FT MEADE, MD 20755

DISTRIBUTION (cont'd)

AMERICAN TELEPHONE & TELEGRAPH CO
ATTN SEC OFC FOR W. EDWARDS
1120 20TH STREET, NW
WASHINGTON, DC 20036

AT&T BELL LABORATORIES
ATTN R. STEVENSON
1600 OSGOOD ST
N. ANDOVER, MA 01845

AT&T BELL LABORATORIES
ATTN J. SERRI
CRAWFORDS CORNER ROAD
HOLMDEL, NJ 07733

BDM CORP
ATTN CORPORATE LIBRARY
7915 JONES BRANCH DRIVE
McLEAN, VA 22102

BOEING CO
PO BOX 3707
ATTN R. SHEPPE
SEATTLE, WA 98124

ENERGISTICS CORP
ATTN R. MANRIQUEZ
1125 JEFFERSON DAVIS HWY
SUITE 1500
ARLINGTON, VA 22202

ENGINEERING SOCIETIES LIBRARY
ATTN ACQUISITIONS DEPT
345 E. 47TH ST
NEW YORK, NY 10017

ENSCO, INC
ATTN R. GRAY
540 PORT ROYAL RD
SPRINGFIELD, VA 22151

IIT RESEARCH INSTITUTE
ATTN J. BRIDGES
ATTN I. MINDEL
10 W 35TH STREET
CHICAGO, IL 60616

INTERNATIONAL TEL & TELEGRAPH CORP
ATTN A. RICHARDSON
ATTN TECHNICAL LIBRARY
500 WASHINGTON AVENUE
NUTLEY, NJ 07110

MARTIN MARIETTA CORPORATION
PO BOX 5837
ATTN DR. C. WHITESCARVER
ORLANDO, FL 32805

MISSION RESEARCH CORP
ATTN TOM BOLT
735 STATE STREET
SANTA BARBARA, CA 93102

MISSION RESEARCH CORP
ATTN W. STARKE
4935 N. 30TH STREET
COLORADO SPRINGS, CO 80933

MISSION RESEARCH CORP
EM SYSTEM APPLICATIONS DIVISION
ATTN A. CHODOROW
1720 RANDOLPH ROAD, SE
ALBUQUERQUE, NM 87106

PRI, INC
ATTN W. HAAS
6121 LINCOLNIA RD
ALEXANDRIA, VA 22312

R&D ASSOCIATES
ATTN W. GRAHAM
PO BOX 9695
MARINA DEL REY, CA 90291

R&D ASSOCIATES
ATTN DIRECTOR, DR. J. THOMPSON
1401 WILSON BLVD
SUITE 500
ARLINGTON, VA 22209

ROCKWELL INTERNATIONAL CORP
ATTN D/243-068, 031-CA31
PO BOX 3105
ANAHEIM, CA 92803

SCIENCE ENGINEERING ASSOC
ATTN P. FLEMMING
ATTN V. JONES
701 DEXTER AVE, N
SEATTLE, WA 98109-4318

SRI INTERNATIONAL
ATTN A. WHITSON
ATTN E. VANCE
333 RAVENSWOOD AVENUE
MENLO PARK, CA 94025

TRW DEFENSE & SPACE SYSTEMS GROUP
ATTN J. PENAR
ONE SPACE PARK
REDONDO BEACH, CA 92078

DISTRIBUTION (cont'd)

TRW DEFENSE & SPACE SYSTEMS GROUP
ATTN E. P. CHIVINGTON
2240 ALAMO, SE
SUITE 200
ALBUQUERQUE, NM 87106

TRW, INC
COMMAND & CONTROL & COMMUNICATIONS
SYSTEM DIV
ATTN N. STAMMER
5203 LEESBURG PIKE
SUITE 310
FALLS CHURCH, VA 22041

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION
ATTN RES & SEC COORD FOR H. DENNY
ATLANTA, GA 30332

US ARMY LABORATORY COMMAND
ATTN TECHNICAL DIRECTOR, AMSLC-TD

INSTALLATION SUPPORT ACTIVITY
ATTN LEGAL OFFICE, SLCIS-CC

USAISC
ATTN RECORD COPY, ASNC-LAB-TS
ATTN TECHNICAL REPORTS BRANCH,
ASNC-LAB-TR (2 COPIES)

HARRY DIAMOND LABORATORIES
ATTN D/DIVISION DIRECTORS
ATTN LIBRARY, SLCIS-IM-TL (3 COPIES)
ATTN LIBRARY, SLCIS-IM-TL (WOODBRIDGE)
ATTN LAB DIRECTOR, SLCHD-NW-E

HARRY DIAMOND LABORATORIES (cont'd)
ATTN CHIEF, SLCHD-NW-EC
ATTN CHIEF, SLCHD-NW-ED (15 COPIES)
ATTN CHIEF, SLCHD-NW-EE
ATTN CHIEF, SLCHD-NW-P
ATTH CHIEF, SLCHD-NW-R
ATTN CHIEF, SLCHD-NW-RA
ATTN CHIEF, SLCHD-NW-RC
ATTN CHIEF, SLCHD-NW-RH
ATTN CHIEF, SLCHD-NW-RI
ATTN CHIEF, SLCHD-TT
ATTN H. LESSER, SLCHD-IT-EB
ATTN J. O. WEDEL, JR., SLCHD-IT-EB
ATTN B. ZABLUDOWSKI, SLCHD-IT-EB
ATTN A. FRYDMAN, SLCHD-IT-RT (2 COPIES)
ATTN R. J. CHASE, SLCHD-NW-EC
ATTN A. HERMANN, SLCHD-NW-EC
ATTN C. LE, SLCHD-NW-EC
ATTN A. NGUYEN, SLCHD-NW-EC
ATTN R. J. REYZER, SLCHD-NW-EC
ATTN D. TROXEL, SLCHD-NW-EC
ATTN R. L. ATKINSON, SLCHD-NW-ED (15 COPIES)
ATTN H. E. BOESCH, JR., SLCHD-NW-RC
ATTN C. FAZI, SLCHD-NW-RE
ATTN R. KAUL, SLCHD-NW-RE
ATTN P. B. JOHNSON, SLCHD-ST-A
ATTN W. WIEBACH, SLCHD-ST-AB
ATTN P. ALEXANDER, SLCHD-ST-AD
ATTN C. ARSEM, SLCHD-ST-AD
ATTN D. M. HULL, SLCHD-ST-AD
ATTN J. LOWE, SLCHD-ST-AD
ATTN J. DENT, SLCHD-TA-F (2 COPIES)
ATTN R. GOODMAN, SLCHD-TA-OS
ATTN J. J. FALTER, SLCHD-NW-ED (50 COPIES)

END

DATE

FILMED

8-88

DTIC